

HOPEX IT Architecture

User Guide

HOPEX V4



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

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

© MEGA International, Paris, 1996 - 2020

All rights reserved.

HOPEX IT Architecture and HOPEX are registered trademarks of MEGA International.

Windows is a registered trademark of Microsoft Corporation.

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

CONTENTS



Contents	1

Introduction to HOPEX IT Architecture V2	15

Presentation of HOPEX IT Architecture V2	17

The Scope Covered by HOPEX IT Architecture V2	17

Summary of Activities and Deliverables of HOPEX IT Architecture V2	18

Structure and positioning of the HOPEX IT Architecture V2 solution	18

HOPEX IT Architecture V2 Profiles	19

Business Roles of HOPEX IT Architecture V2	20

The HOPEX IT Architecture V2 method	21

Building the Business Capability Maps of the Organization	21

<i>Describing Business Capabilities</i>	21

<i>Identifying the functionalities associated with business capabilities</i>	22

<i>Identifying the applications associated with functionalities</i>	25

Building the Logical Architecture	25

<i>Structure diagram of the logical application system</i>	27

<i>Logical application system environment diagram</i>	28

Describing Application Architecture	29

<i>Describing application systems</i>	30

<i>Describing application processes</i>	32

<i>Describing flow scenarios</i>	33

Defining the application technical architecture	34

Analyzing the functional coverage of the architecture implemented	35

<i>Analyzing the functional coverage of the application architecture</i>	35

Describing batch processing	36

Describing the list of services and interfaces	36

Defining the technical infrastructure	37

<i>Resource Architecture Environment Diagram</i>	37

<i>Describing Resource Architectures</i>	38

<i>IT infrastructure assembly structure diagram</i>	39

<i>Computing Device Assembly Diagram</i>	41

Managing Service Catalogs	41

HOPEX IT Architecture V2 Desktop Presentation	43
Connecting to the solution	43
HOPEX IT Architecture V2 Desktop Presentation	43
<i>Presentation of space common to all profiles</i>	<i>44</i>
<i>Presentation of the IT Architecture Functional Administrator space</i>	<i>45</i>
<i>Presenting the IT Architecture Manager space</i>	<i>48</i>
<i>Presentation of the Application Architect space</i>	<i>49</i>
<i>Presentation of the Infrastructure Architect space</i>	<i>50</i>
<i>Presentation of the IT Technical Architect space</i>	<i>51</i>
<i>Presentation of the Application Contributor space</i>	<i>52</i>
<i>Presentation of the Application Viewer space</i>	<i>52</i>
<i>Presentation of the UML Designer desktop</i>	<i>53</i>
Switching between Profiles	54
Before starting with HOPEX IT Architecture V2	55
Using conversion Tools to HOPEX V4	55
Preparing the Work Environment	56
<i>Accessing the list of libraries with HOPEX IT Architecture V2</i>	<i>56</i>
<i>Creating a library with HOPEX IT Architecture V2</i>	<i>56</i>
Using properties pages	56
<i>Displaying the properties window on a permanent basis</i>	<i>56</i>
<i>HOPEX IT Architecture V2 properties pages content</i>	<i>57</i>
Using Org-units	58
<i>Creating an org-unit</i>	<i>58</i>
<i>Internal org-unit/external entity</i>	<i>59</i>
Importing components with HOPEX IT Architecture V2	59
<i>Structure of the import/export Excel templates of HOPEX IT Architecture V2</i>	<i>59</i>
<i>Importing computing devices or technologies with Excel</i>	<i>61</i>
<i>Building the import file for HOPEX IT Architecture V2</i>	<i>62</i>
About This Guide	66
Guide Structure	66
Additional Resources	66
Conventions used in the guide	67

Modeling Business Capabilities and the logical application architecture 71

Using Business Capabilities with HOPEX IT Architecture V2	72
Business capabilities examples with HOPEX IT Architecture V2	72
Managing the Business Capability Maps with HOPEX IT Architecture V2	73
<i>Accessing the list of business capability maps</i>	<i>73</i>
<i>Creating a business capability map</i>	<i>73</i>
<i>Creating a business capability map diagram</i>	<i>73</i>
<i>The properties of a business capability map</i>	<i>74</i>
Managing Business Capabilities with HOPEX IT Architecture V2	74
<i>Describing a business capability</i>	<i>75</i>
<i>Accessing the list of business capabilities with HOPEX IT Architecture V2</i>	<i>75</i>
<i>Defining the functionalities associated with business capabilities</i>	<i>75</i>
Using Functionalities with HOPEX IT Architecture V2	77
Describing a Functionality Map with HOPEX IT Architecture V2	77
<i>Accessing the list of functionality maps with HOPEX IT Architecture V2</i>	<i>78</i>

<i>The properties of a functionality map</i>	78
<i>Creating a functionality map</i>	78
Describing functionalities with HOPEX IT Architecture V2	79
<i>Creating a Functionality Diagram with HOPEX IT Architecture V2</i>	79
Describing a Technical Functionality	79
Describing a hardware functionality	80
Describing a Logical Application Architecture with HOPEX IT Architecture V2	81
Describing a Logical Application System with HOPEX IT Architecture V2	81
<i>Accessing the list of logical application systems with HOPEX IT Architecture V2</i>	81
<i>Creating a Logical Application System</i>	81
<i>Logical Application System Properties</i>	81
<i>Describing a Logical Application System with HOPEX IT Architecture V2</i>	82
Describing Logical Applications With HOPEX IT Architecture V2	84
<i>Accessing the list of logical applications with HOPEX IT Architecture V2</i>	84
<i>Creating a logical application</i>	84
<i>Logical Application Properties</i>	85
Logical Application System Environment Description.	85
<i>Example of logical application system environment</i>	86
<i>Accessing the list of logical application system environments</i>	86
<i>Creating a logical application system environment</i>	86
<i>Logical application system environment properties</i>	87
<i>Using the Environment Structure Diagram of a Logical Application System</i>	87
Using Implementation mechanisms	89
Describing Fulfillment of a Business Capability.	89
<i>Creating Fulfillment of a Business capability</i>	89
<i>Analyzing enterprise capability implementation</i>	89
Describing the fulfillment of a Functionality	90
<i>Creating Fulfillment of a Functionality</i>	90
<i>Identifying the logical applications associated with Functionalities</i>	90

Modeling Technical and Functional Architectures 93

Describing Application Architecture	94
HOPEX IT Architecture V2 Concepts Overview	94
<i>Application</i>	94
<i>Application System</i>	94
Describing an Application with HOPEX IT Architecture V2	95
<i>Creating an Application with HOPEX IT Architecture V2</i>	95
<i>The properties of an application with HOPEX IT Architecture V2</i>	96
<i>Defining Application Functional Scope</i>	97
<i>Specifying the Risks Associated with an Application</i>	97
Describing an Application Environment with HOPEX IT Architecture V2	98
<i>Accessing the List of Application Environments</i>	98
<i>Creating an application environment</i>	98
<i>Application environment properties</i>	98
<i>Creating an application environment use case diagram</i>	100
Describing an Application System.	100
<i>Creating an Application System</i>	101
<i>Application System Properties</i>	101

<i>Creating an application system structure diagram</i>	103
Describing an Application System Environment with HOPEX IT Architecture V2	105
<i>Accessing the list of application system environments.</i>	105
<i>Creating an application system environment</i>	105
<i>Application system environment properties</i>	105
<i>Describing an application system environment diagram</i>	105
<i>Creating an Application Environment Use Case Diagram</i>	107
Describing Application Flows	109
Creating a Global Application Flow Map	109
<i>Accessing the list of global application flow maps</i>	109
<i>The properties of a global application flow map</i>	109
Using a Scenario of Application System Flows.	109
<i>Adding an org-unit to the Scenario of Application System Flows.</i>	111
<i>Adding a scenario of application system flows</i>	111
Describing a Scenario of Application System Environment	112
Managing Data	113
Using Data Stores.	113
<i>Introduction to data store concept</i>	113
<i>Accessing to data areas with HOPEX IT Architecture V2.</i>	113
<i>Creating a data area.</i>	113
Using Data Stores.	114
<i>Introduction to the data store concept</i>	114
<i>Usage contexts</i>	114
<i>Creating a local data store.</i>	115
<i>Creating a external data store</i>	115
<i>Describing access to a data store</i>	116
UML modeling of data	116
<i>UML package</i>	117
<i>Data models</i>	117
<i>Data areas.</i>	118
Describing Application Processes	120
Managing System Processes with HOPEX IT Architecture V2	120
<i>Accessing system processes</i>	121
<i>Creating a system process diagram</i>	121
Specifying the behavior of a task in a System Process	123
<i>Behaviors</i>	124
<i>Task type</i>	124
Describing the tasks of a System Process	125
<i>Functional Modeling Example</i>	125
<i>Display the diagram describing a step in the system process in detail:</i>	125
Modeling Tasks of an IT Service	127
Describing Technical Architecture	128
Describing a Software Technology.	128
<i>Accessing the list of software technologies</i>	128
<i>The properties of a software technology</i>	128
Describing Software Technology Stacks.	129
<i>Accessing the list of technology stacks</i>	129
<i>Properties of a software technology stack</i>	129
Creating an Application Technical Architecture	129
<i>Accessing the application technical architectures</i>	131
<i>Properties of an application technical architecture</i>	131
<i>Creating an Application Technical Architecture Diagram</i>	131

Describe an application system technical architecture.	133
<i>Accessing the list of application system technical architectures</i>	134
<i>Properties of an application system technical architecture.</i>	134
<i>Creating an Application System Technical Architecture Diagram</i>	135
Describing Batch Processing	136
Defining a Batch Process	136
Building a Batch Planning Structure Diagram	136
<i>Creating a batch planning structure diagram.</i>	137
<i>Adding a call for batch processing in the diagram</i>	137
<i>Defining batch sequencing</i>	138
Creating a Batch Program Structure Diagram	138
<i>Creating a batch program structure diagram.</i>	138
<i>Adding a programming call to the diagram</i>	138
Using system process batch realizations	139
Defining User Interfaces	140
Creating a user interface	140
Building a User Interface Diagram	140
Drawing the Interface Diagram	141
<i>User interface element.</i>	141
<i>User interface event.</i>	142
 Modeling IT Infrastructures	 145
Describing a Resource Architecture	146
Creating a Resource Architecture	146
Managing a Resource Architecture Assembly Diagram	146
<i>Creating a Resource Architecture Assembly Diagram:</i>	146
<i>Adding a Resource Architecture.</i>	146
<i>Adding an Org-Unit or a Position Type</i>	147
<i>Describing Resource Architecture Technical Resources</i>	147
<i>Describing the Services in a Resource Architecture Assembly Diagram.</i>	148
Describing a Resource Architecture Environment	149
<i>Creating a resource architecture environment</i>	149
<i>To create a resource architecture environment diagram</i>	149
Describing IT Components	151
Describing an IT infrastructure.	151
<i>Creating an IT infrastructure.</i>	151
<i>Creating an Infrastructure Assembly Structure Diagram.</i>	151
<i>Using an Infrastructure Assembly Structure Diagram.</i>	151
Describing a Computing Device	152
<i>Accessing the list of computing devices</i>	152
<i>Creating a computing device.</i>	152
<i>Creating a Computing Device Assembly Diagram.</i>	153
Describing an IT network	154
<i>Creating an IT network.</i>	154
<i>Creating an IT network.</i>	154
Describing communications in a Technical Infrastructure.	156
Describing services and requests	156
<i>Service points</i>	156

<i>Request points</i>	156
Describing Communications at Equipment Level	157
<i>Communication ports</i>	157
<i>Communication channels</i>	157
<i>Network communication protocols</i>	158
Using service catalogs	161
Introducing service catalogs	162
The types of service catalogs	162
Implementation of a service catalog	163
Populating a service catalog	165
<i>Creating an information service catalog</i>	165
<i>Accessing the list of service catalogs</i>	166
<i>Adding a service catalog item</i>	166
<i>Specifying the implementation of a service catalog item</i>	166
Service catalog reports	168
Using IT architecture diagrams	169
Structure diagrams and flow scenario diagrams initialization	170
Creating a structure diagram	172
Creating an Application Structure Diagram	172
<i>The components of an Application Structure Diagram</i>	172
<i>Adding an application service to an application structure diagram</i>	173
Describing a scenario of flows	174
Using a Scenario of Application Flows Diagram	174
<i>Creating a Scenario of Application Flows diagram</i>	175
<i>Adding an IT service to the scenario of application flows</i>	175
<i>Managing application flows in a scenario of application flows</i>	175
<i>Adding an application data store to the scenario of application system flows</i>	176
<i>Creating an application data channel</i>	177
Using a Flow Scenario Sequence Diagram	177
<i>Using a Flow Scenario Sequence Diagram</i>	178
<i>Instances of applications, IT services or interfaces</i>	178
<i>Message instance</i>	179
Describing a Technical Architecture	180
Creating an Application Technical Architecture Diagram	180
<i>Adding an application technical area to an application technical architecture diagram</i>	181
<i>Defining the software technologies used by an application technical area</i>	181
<i>Describing technical communications</i>	182

Describing data exchanges. 185

Managing Interactions 186

Creating an Interaction	187
Describing Service and Request Points	187
<i>Service points</i>	187
<i>Request points</i>	188
<i>Creating a Service Point or a Request Point.</i>	189
Defining the Element Interaction Point	189
<i>Characterizing the element interaction point</i>	189

Describing Exchanges. 191

Creating an Exchange.	191
Describing Exchanges	192
<i>Creating an exchange diagram (BPMN)</i>	192
<i>Creating a message flow with content</i>	192
<i>Managing events, gateways and sequence flows</i>	193

Describing Exchange Contracts 194

Examples of Exchange Contract Diagrams (BPMN)	195
<i>Exchange contract diagram (BPMN)</i>	195
<i>Advanced communication exchange contract example</i>	195
Creating an Exchange Contract from an Interaction	196
Creating an Exchange Contract Diagram (BPMN)	197
Defining an Exchange or an Exchange Contract Use	197

HOPEX IT Architecture V2 Reports. 199

Business Capabilities reports 200

Breakdown map of business capabilities	200
Business Capabilities and Expected Functionalities Matrix	201

Reports on the Architecture Functional Coverage 203

Building Block Breakdown report	203
Matrix of Business Capabilities and Expected Functionalities	206

Application Architecture Reports 208

Application Architecture Breakdown Report	208
Impact report (Scenario).	210
Impact report (Structure)	212

Infrastructure Reports 214

Infrastructure Description Report	214

About UML implementation 217

Overview. 218

<i>Analyzing use cases</i>	218
<i>Identifying objects</i>	218
<i>Describing behaviors</i>	218

<i>Representing interactions between objects</i>	218
<i>Dividing classes between packages</i>	219
<i>Defining interfaces</i>	219
<i>Specifying deployment</i>	219
Organization of UML Diagrams	220
<i>General organization</i>	220
<i>Detailed specification</i>	220
<i>Technical specification and deployment</i>	221
<i>UML diagram entry points</i>	221

Use Case Diagram 223

Creating a Use Case Diagram	224
Creating a Package	224
Creating the Use Case Diagram of a Package	224
Use Case Diagram Elements	225
Actors	225
Use Cases	226
<i>Zooming in on a use case</i>	226
Packages	226
Participations	227
<i>Examples of participation</i>	228
<i>Creating participations</i>	228
<i>Multiplicities of a participation</i>	229
Use Case Associations: Extensions and Uses	229
<i>Inclusion relationship</i>	229
<i>Extend Relation</i>	230
Generalizations	232
Interfaces	233
<i>Creating an Interface</i>	233
<i>Connecting an interface to a use case</i>	233

The Class Diagram 235

Presentation of the Class Diagram	236
<i>The Class Diagram: summary</i>	237
Creating a Class Diagram	237
Classes	238
Definition: Class	238
Creating a Class	239
<i>Finding an existing class</i>	239
Class Properties	239
<i>Class characteristics page</i>	240
<i>Other properties pages</i>	241
Class Stereotype	241
<i>Stereotype display option</i>	242

Attributes	243
Definition: Attribute	243
Specifying Class Attributes	244
<i>Creating a standard attribute</i>	244
<i>Creating a computed attribute</i>	244
<i>Inherited attributes</i>	245
Attribute Properties	245
<i>Attribute type</i>	245
Operations	247
Definition of an Operation	247
Specifying Class Operations	247
<i>Inherited operations</i>	247
Operation Properties	248
Operation or Signal Signatures	248
<i>Signature syntax</i>	249
Operation Parameters	249
Operation Methods (opaque behavior)	250
Operation Conditions	250
<i>Operation Exceptions</i>	251
Displaying Class Attributes and Operations	251
Signals	252
Defining a Signal	252
Specifying Class Signals	252
<i>Creating a sent or received signal</i>	252
<i>Signal Properties</i>	252
<i>Signal parameters</i>	253
Associations	254
Creating an Association	255
Roles (or Association Ends)	255
Multiplicity of a Role	256
<i>Specifying role multiplicity</i>	258
Association End Navigability	259
<i>Specifying navigability for a role</i>	260
Association End Aggregation	260
<i>Specifying role aggregation</i>	260
Association End Composition	260
Role Changeability	261
Role Order	261
Role Static Property	262
Role Qualifier	262
Overloading a Role	262
Association Classes	263
Displaying an N-ary Association	264
Reflexive Associations	264
<i>Creating a reflexive association</i>	265
The Parts	266
Creating a Part between two Classes	266
Defining the Identifier of a Class via a Part	266
Multiplicities of the Associated Classes	267
<i>Multiplicity of the class referenced by the part</i>	267
<i>Multiplicity of the owner class of the part</i>	268
Aggregation and Composition Relationships	268

<i>Associated multiplicities</i>	269
Generalizations	270
What is a Generalization?	270
<i>Example</i>	271
Multiple Subclasses - Generalization	272
Advantages of Subclasses - Generalization	272
Multiple Inheritance - Generalization	273
Creating a generalization	273
Discriminator - Generalization	274
Specifying Interfaces	275
Creating an Interface	275
<i>Connecting an interface to a class</i>	275
Specifying Dependencies	276
Specifying Parameterized Classes	277
Constraints	278
Object Diagram	279
Objects	279
<i>Creating an object (instance)</i>	280
<i>Instance properties</i>	280
<i>Value of an attribute</i>	281
Links	281
<i>Creating a link</i>	282
<i>Link properties</i>	282
<i>Role properties</i>	282
Structure and Deployment Diagrams	285
The Package Diagram	286
Creating a Package Diagram	286
Defining Packages	287
Defining Classes	287
Specifying Dependencies in a Package Diagram	287
The Component Diagram	289
Creating a Component Diagram	289
Components	290
Interfaces	290
<i>Creating component interfaces</i>	290
<i>Linking interfaces to other objects</i>	290
<i>Connecting interfaces</i>	291
Ports	291
Connectors	291
<i>Delegate connector</i>	292
<i>Assembly connector</i>	292
Composite Structure Diagram	293
Creating a Composite Structure Diagram	293
Parts	294
Collaborations	294
<i>Collaboration use</i>	295

<i>Collaboration use example</i>	295
Dependency links	295

State Machine Diagram 297

Presentation of the State Machine Diagram	298
Creating a State Machine Diagram	298
States	300
Creating a State	300
<i>State types</i>	300
<i>Pseudo-states</i>	301
Detailing Behavior of a State	302
State Properties	303
State Transitions	304
Creating a Transition	304
Transition Types	304
<i>External transition</i>	304
<i>Internal transition</i>	304
<i>Local transition</i>	305
Transition Effects	305
<i>Transition Effect Display</i>	305
Transition Triggering Event	305

Activity Diagram 307

Activity Diagram	308
Creating an Activity Diagram	308
Partitions	309
Creating a Partition	309
Partition Properties	309
Nodes	311
Object nodes	311
<i>Creating an Action</i>	311
<i>Modifying the Action Type</i>	311
Parameter nodes	311
Control nodes	312
<i>Control node types</i>	312
Object nodes: Input, Output and Exchange Pins	313
<i>Input pin</i>	313
<i>Output pin</i>	313
<i>Exchange pin</i>	313
Flows	313
<i>Control flow</i>	313
<i>Object flows</i>	313

Interaction Diagrams	315
Interactions.	316
Creating an Interaction	316
Creating an Interaction Diagram	316
Sequence Diagram	317
Creating a Sequence Diagram	318
Lifelines	318
<i>Creating a lifeline</i>	<i>318</i>
<i>Lifeline properties.</i>	<i>318</i>
Messages	318
<i>Examples of exchanged messages</i>	<i>319</i>
<i>Creating a message</i>	<i>319</i>
<i>Message types</i>	<i>320</i>
Execution Specification	320
<i>Creating an execution specification</i>	<i>320</i>
Occurrence specification	320
<i>Calculating sequence numbers.</i>	<i>321</i>
Combined Fragment	322
<i>Creating a combined fragment.</i>	<i>322</i>
<i>Interaction operator type</i>	<i>324</i>
<i>Interaction operands.</i>	<i>325</i>
Interaction Use	326
Gate	327
Continuation	327
Communication Diagram.	328
<i>Example</i>	<i>328</i>
<i>Diagram objects.</i>	<i>329</i>
Interaction Overview Diagram	330

The deployment diagram	331
Presentation of the Deployment Diagram.	332
Creating a Deployment Diagram	332
Deployment Diagram Objects	333
<i>Node.</i>	<i>333</i>
<i>Communication path.</i>	<i>333</i>
<i>Component</i>	<i>333</i>
<i>Artifact</i>	<i>333</i>
<i>Manifestation.</i>	<i>333</i>
<i>Deployment specification.</i>	<i>334</i>
<i>Configuration.</i>	<i>334</i>

Appendix: Attribute type	335
Primitive Types	336
Prerequisite: Importing the Primitive Types	336
Defining a Primitive Type	336
Packages and Primitive Types	338
Packages	338
Defining New Primitive Types	341
Compound Primitive Type	342

INTRODUCTION



HOPEX IT Architecture V2 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 V2 offers facilities for different analysis perspectives:

- 6 **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.
- 6 **Application mapping:** a description of application architecture that offers a detailed view of information exchanges between applications, services, databases and organizational units.
- 6 **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.
- 6 **The representation of resource architectures:** a description of complex systems involving different types of IT resources.

In addition to **HOPEX IT Architecture V2**, **HOPEX IT Portfolio 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 V2 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 V2**. 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 V2**:

- 6 [Modeling Business Capabilities and the logical application architecture;](#)
- 6 [Modeling Technical and Functional Architectures;](#)
- 6 [Modeling IT Infrastructures;](#)
- 6 [Using service catalogs;](#)
- 6 [Describing data exchanges;](#)
- 6 [About UML implementation.](#)

PRESENTATION OF HOPEX IT ARCHITECTURE V2

Combined with the products of the **HOPEX** suite, **HOPEX IT Architecture V2** supports a methodology and the tools used to describe, analyze and plan your information system transformation.

The Scope Covered by HOPEX IT Architecture V2

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 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 [Building the Business Capability Maps of the Organization](#) and [Building the Logical Architecture](#).*

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

- *For more details, see [Describing Application Architecture](#).*

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 V2** 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 [Defining the application technical architecture](#).*

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

- *For more details, see [Analyzing the functional coverage of the architecture implemented](#).*

Describing the service catalogs: this last step consists in grouping in a service catalog the list of functionalities covered by the systems described.

- *For more details, see [Managing Service Catalogs](#).*

Using UML formalism : at least, you can use the UML (Unified Modeling Language) modeling language to model your IS.

- For more details, see [About UML implementation](#).

Summary of Activities and Deliverables of HOPEX IT Architecture V2

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 V2** 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 a Logical Application Architecture with HOPEX IT Architecture V2 .
Building the application architecture	Application architecture structure diagrams and flow scenario diagrams, see Describing Application Architecture and Describing Application Flows .
Analyzing the functional coverage of the application architecture	Assessing the functional coverage by software resources, see Describing the fulfillment of a Functionality . Assessment of the coverage of technical functionalities by technical resources, see Describing the fulfillment of a Functionality .
Defining the technical architecture	Description of the technical requirements for the application deployment, see Describing Technical Architecture .
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 V2 deliverables

Structure and positioning of the HOPEX IT Architecture V2 solution

HOPEX IT Architecture V2 can be used with other products in the **HOPEX** suite.

HOPEX IT Portfolio Management

The **HOPEX IT Portfolio Management** solution provides **HOPEX IT Architecture V2** with the possibility to define whether the business capabilities are implemented

by the applications. Both solutions share the mapping functionality for business capabilities.

HOPEX IT Business Management

In addition to **HOPEX IT Portfolio Management**, **HOPEX IT Business Management** solution provides the possibility to describe the organizations and processes that implement the business capabilities identified in **HOPEX IT Architecture V2**;

HOPEX Risk Mapper

The **HOPEX Risk Mapper** solution provides **HOPEX IT Architecture V2** with the possibility to associate risks with applications and application architecture elements.

HOPEX IT Architecture V2 Profiles

In **HOPEX IT Architecture V2**, 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 V2 Desktop Presentation](#).

Profiles	Tasks
IT Architecture Functional Administrator	In addition to the functional rights of the application manager, the IT Architecture Functional Administrator has rights to all objects, methods, projects and workflows. He/she prepares the work environment and creates the elements required to manage the modeled elements. He manages: - Users, assignment of roles and access rights to the different project steps. - all the environment objects (application environments, infrastructures, reports, etc.), - Workflows. For more details, see Presentation of the IT Architecture Functional Administrator space .
IT Architecture Manager	The IT Architecture Manager has rights to all objects, methods, projects and assessments. For more details, see Presenting the IT Architecture Manager space .
Application architect	The application architect is responsible for building the application architecture models assigned to him/her. He/she can manage transformation projects. For more details, see Presentation of the Application Architect space .

Profiles	Tasks
IT infrastructure architect	The IT infrastructure architect is responsible for building the infrastructure and hardware models assigned to him/her. He/she can manage transformation projects. For more details, see Presentation of the Infrastructure Architect space .
IT Technical Architect	The IT Technical Architect is responsible for building the technical architecture models assigned to him/her. He/she can manage transformation projects. For more details, see Presentation of the IT Technical Architect space .
the UML designer,	The UMLdesigner is responsible for the detailed specifications of the system and for building UML diagrams. For more details on the UML designer, see Presentation of the UML Designer desktop .
Applications Contributor and Application Contributor (lite)	The applications contributor is responsible for validating the design of the applications assigned to him/her. For more details, see Presentation of the Application Contributor space .
Application Viewer and Application Viewer Lite,	Applications viewer and Applications Viewer (lite) have read-only rights on objects in the repository. For more details, see Presentation of the Application Viewer space .

Business Roles of HOPEX IT Architecture V2

In **HOPEX IT Architecture V2**, 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.
- **Local Application Owner** used to assign a user to applications. The Local Application Owner is responsible for the following tasks:
 - Identifies risks
 - Responding to Questionnaires
 - Defining and implementing action plans,
 - Validating the modifications made by the architect in the context of object review workflows.
- **IT Owner**: specifies the characteristics of IT elements for which he/she is responsible.
- **Business Owner**: specifies the characteristics of software and applications for which he/she is responsible.
- **Data Designer** , who is responsible for managing data.

THE HOPEX IT ARCHITECTURE V2 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.

Building the Business Capability Maps of the Organization

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

- For more details on managing business capabilities with **HOPEX IT Architecture V2**, see [Using Business Capabilities with HOPEX IT Architecture V2](#).

Describing Business Capabilities

A **business capability** defines an expected skill.

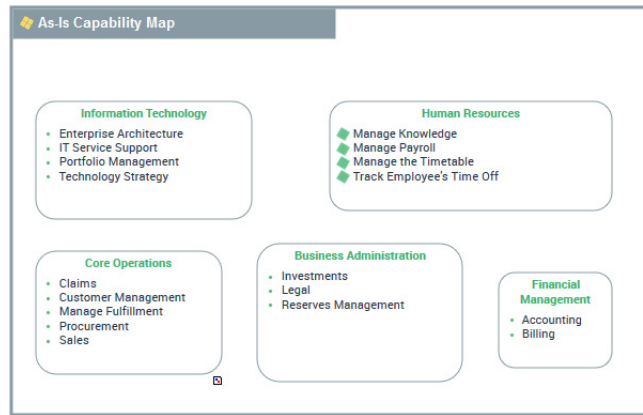
) *.A business capability is a set of features that can be made available by a system (an enterprise or an automated system).*

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 that, together, 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 [Using Business Capabilities with HOPEX IT Architecture V2](#).

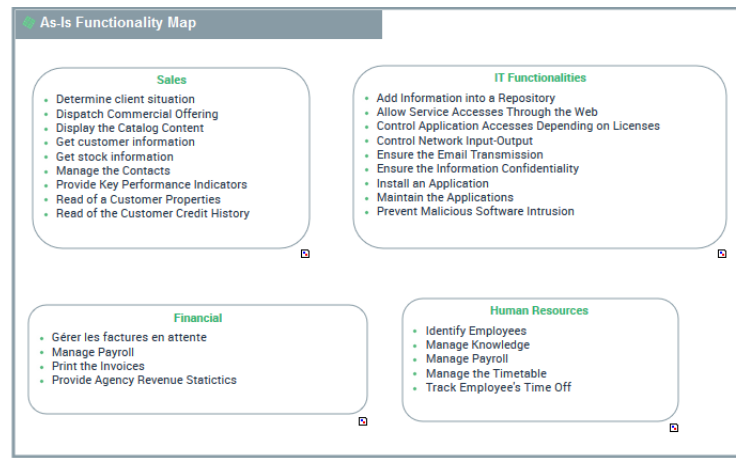
Identifying the functionalities associated with business capabilities

A **functionality** is an aptitude expected from an equipment.

) A functionality is a service required by an org-unit in order to perform its 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.

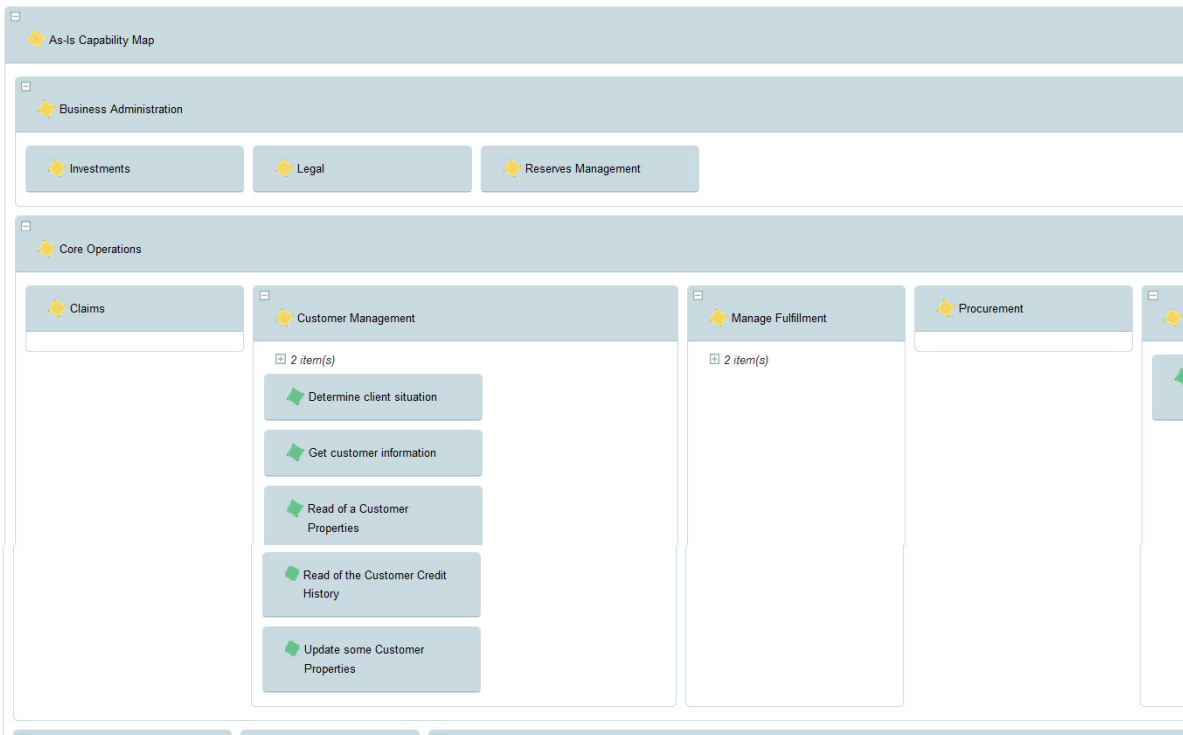
) A functionality map is a set of functionalities with their dependencies that, jointly, define the scope of a hardware or software architecture.



Example of a functionality map

- For more details on managing a functionality map, see [Describing a Functionality Map with HOPEX IT Architecture V2](#).

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



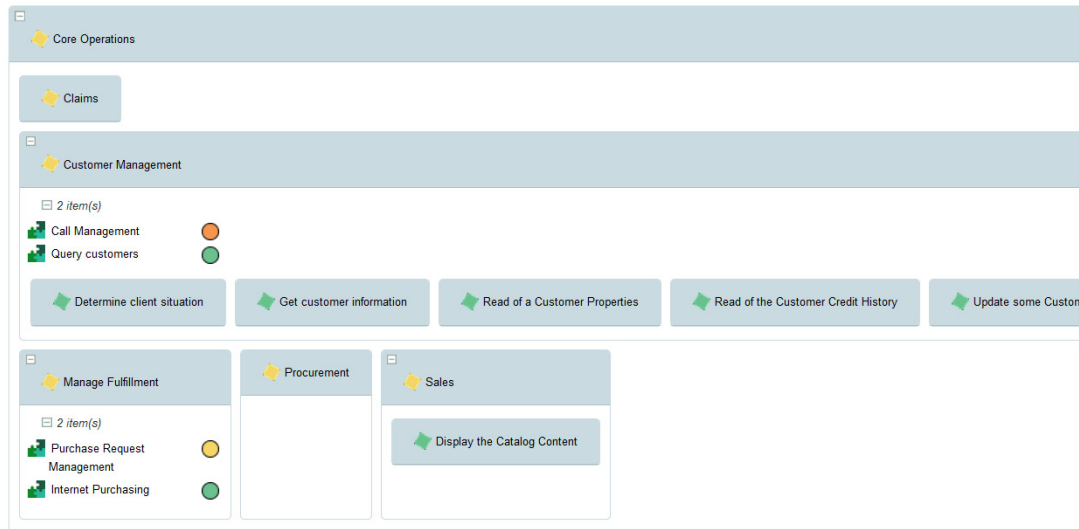
Example of a business capability breakdown report

- For more details on this breakdown report, see [Describing Fulfillment of a Business Capability](#).

Identifying the applications associated with functionalities

Applications cover *functionalities* associated with *business capabilities*. In **HOPEX IT Architecture V2**, a report allows to check the functional coverage of your applications.

- For more details on this breakdown report, see [Describing Fulfillment of a Business Capability](#).



Building the Logical Architecture

HOPEX IT Architecture V2 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 a Logical Application Architecture with HOPEX IT Architecture V2](#).

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 will need to describe the *logical application system environment* to describe the context in which the logical system is used and its interactions with external logical components.

) *A logical application system environment presents a logical application system use context. It describes the 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:

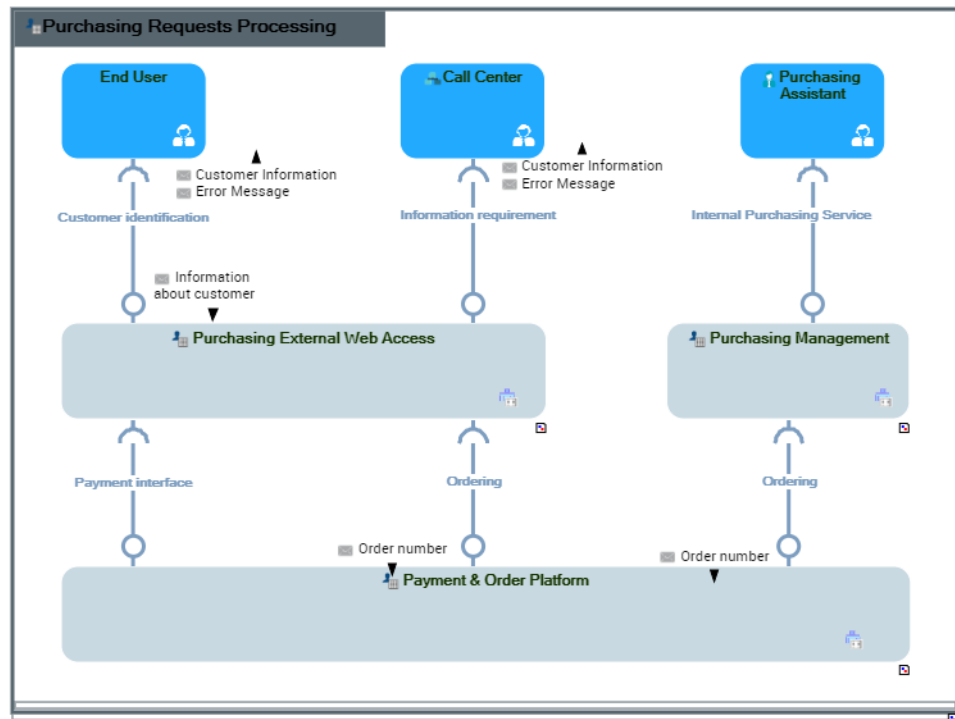
- Identify the exchanges between the different logical components and formalize them through *exchange contracts*.
 -) *An exchange contract is a model of a contract between organizational entities. This contract is described by exchanges between an initiator role and one or several contributor roles.*
 - *For more details on exchange contracts, see [Describing data exchanges](#).*
- Verify that the logical architecture covers the functional requirements identified in the business capability maps.

Structure diagram of the logical application system

The components of the *logical application system* are described in a structure diagram of the logical application system that presents:

- the services offered or required;
- the processes handled, components and their interactions;
- 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.



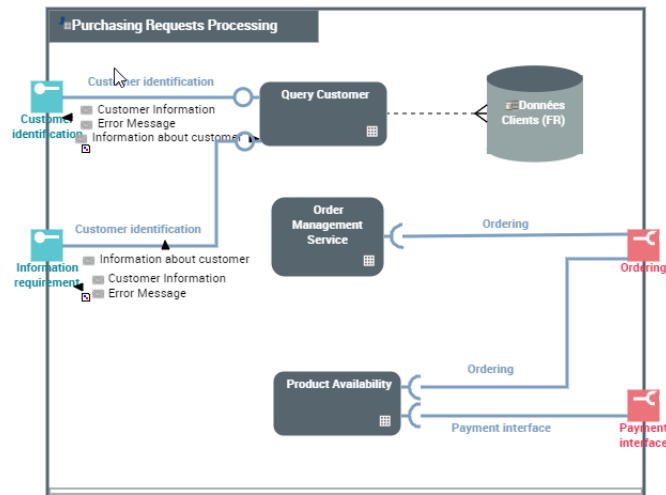
Structure diagram of the "Internet Purchasing" logical application system

"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 "Internet Purchase Requests", presents different

logical applications, access to a logical database as well as service and request points for "Book" or "Order".



Structure diagram of the "Management Shopping Cart" logical application system

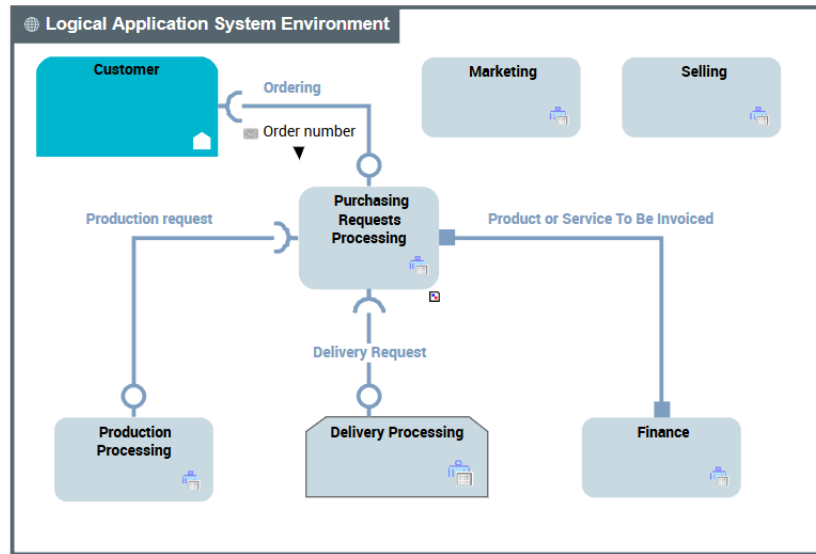
- For more details on use of a logical application system, see [Describing a Logical Application System with HOPEX IT Architecture V2](#).

Logical application system environment diagram

) A logical application system environment presents a logical application system use context. It describes the 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

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.

- For more details on use of a logical application environment system, see [Logical Application System Environment Description](#).

Describing Application Architecture

HOPEX IT Architecture V2 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 details on the use of an application architecture, see [Modeling Technical and Functional 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.*

If you use a unitary approach, you will need to describe the *application system environment* to describe the context in which the application system is used and its interactions with external components.

) *An application system environment allows presenting the other application systems, applications or micro-services with which this application system can interact.*

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

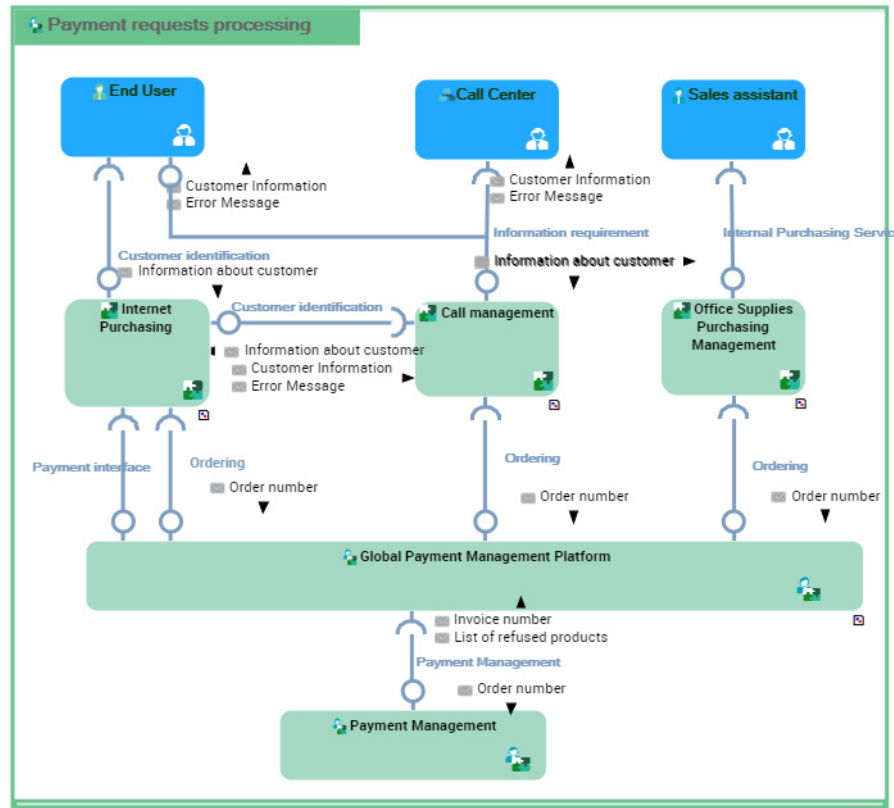
In an approach where application systems are studied in a unitary manner, it is preferable to create application system environments to describe the context in which the application systems described are used.

- *For more details on application system environments, see [Describing an Application System Environment with HOPEX IT Architecture V2](#).*

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.

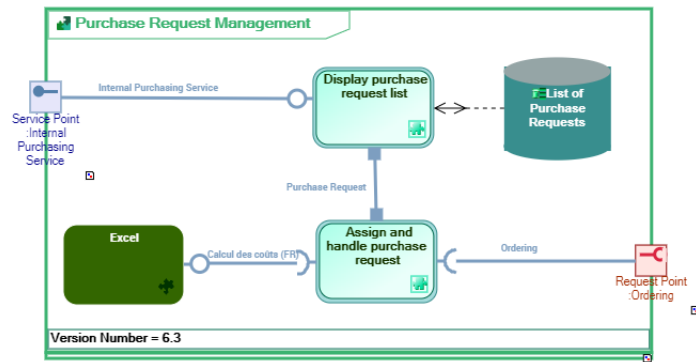
The application subsystems can then be described hierarchically by showing at each level the points of exchange with the outside world.

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](#).



Application sub-system structure diagram of "Purchasing Request Management".

The "Purchase Request Management" application uses two IT Services: "Display purchase request list" and "Assign and handle purchase request". The "Assign and handle purchase request" IT service invokes Excel micro-service.

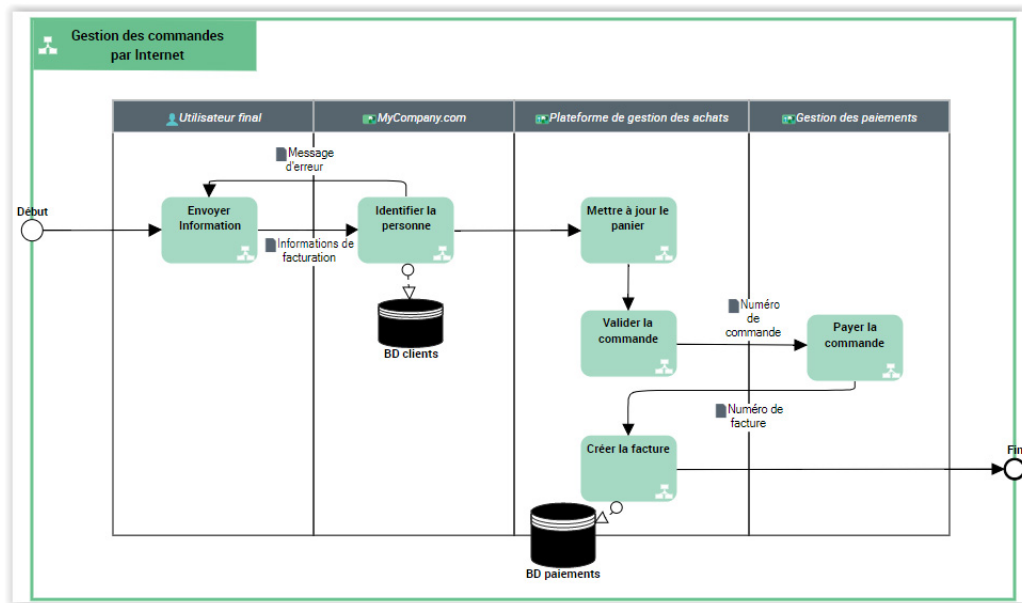
Describing application processes

HOPEX IT Architecture V2 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 Application Processes](#).



Application process diagram

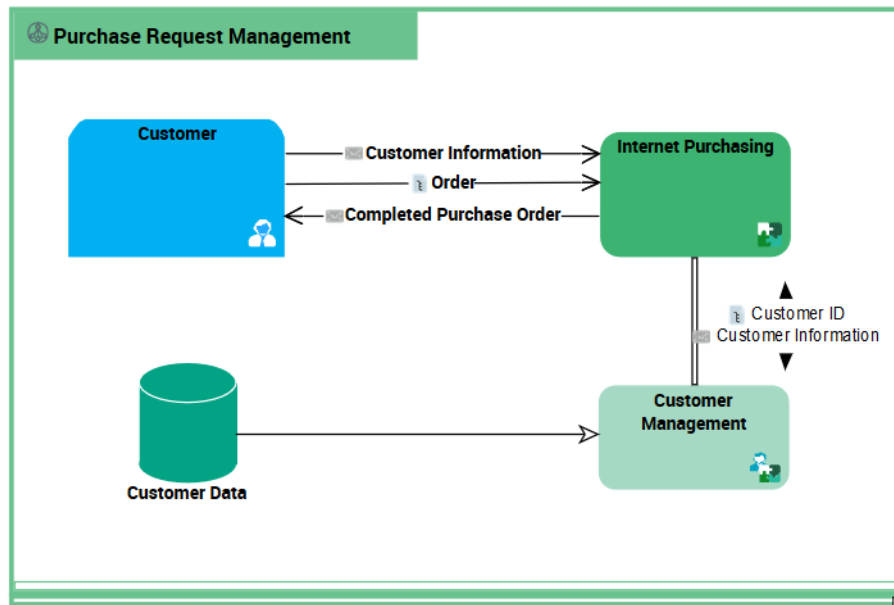
Describing flow scenarios

Structure diagrams use interactions to describe the data exchanged between components. However **HOPEX IT Architecture V2** 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 system flows below describes the interactions between a client and the eCommerce application.



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

- For more details on flow scenarios, see [Describing Application Flows](#).

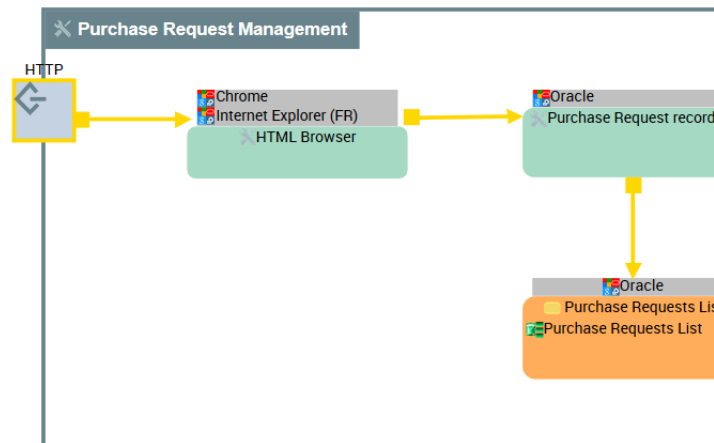
Defining the application technical architecture

An application technical architecture allows to represent packages to be deployed in the form of *Application Technical Area* and *Technical Data Area* as well as the *Technical Communication Line* necessary for connections between areas (protocols).

-) An application technical area represents an organizational element of an application according to technical criteria. For example, this can be the user interface or a process. Each application technical area is associated with one or more technologies. The deployment of several application technical areas is necessary for the application to be operation.
-) A data technical area represents an organizational element of an application used to access the data necessary for the operation of this application. Each application technical area is associated with one or more technologies (E.g.: Oracle 12, SQL Server 2012, etc.). A data technical area can allow access to one or more data stores.
-) A technical communication line represents a technical connection between architectures or application technical areas through client and

server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.

The application technical architecture below presents the technical application areas used by the "Purchase Request Management" application. The technical communication line between the communication port and the "HTML Browser" is based on the HTTP communication protocol.



Example of the application technical architecture for "Purchasing Request Processing".

- For more details, see [Describing Technical Architecture](#).

Analyzing the functional coverage of the architecture implemented

The goal of this step is to check the adequacy between the expected functionalities and the architecture elements that deliver them.

Analyzing the functional coverage of the application architecture

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* at 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.

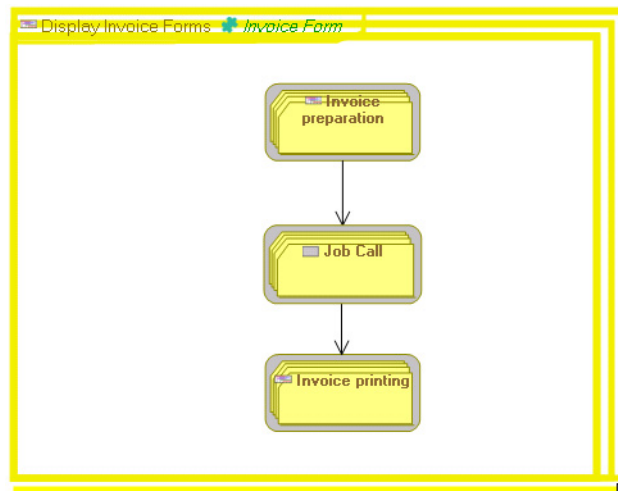
- For more details on the logical applications associated with business capabilities, see [Describing the fulfillment of a Functionality](#).

HOPEX IT Architecture V2 provides a report that presents the result of the implementation of business capabilities by *Logical Application* or *Logical Application System*.

- For more details on the breakdown of business capabilities, see [Breakdown map of business capabilities](#).

Describing batch processing

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

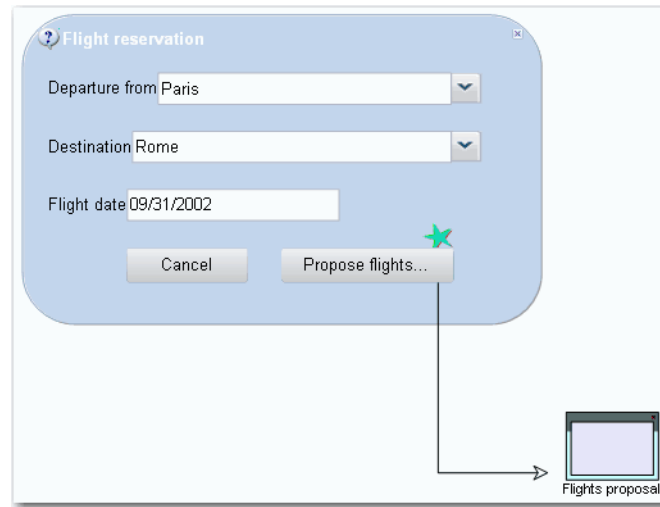


- For more details on describing batch processing, see [Describing Batch Processing](#).

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](#).

Defining the technical infrastructure

Describing the technical infrastructure helps to design applications and data technical areas in preparing their deployment.

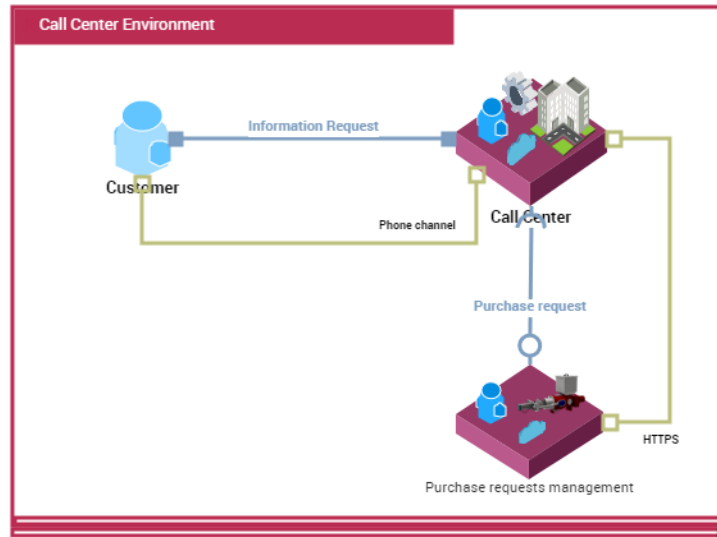
Technical infrastructure elements are identified and characterized by technologies and hosted IT services.

With **HOPEX IT Architecture V2** 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.

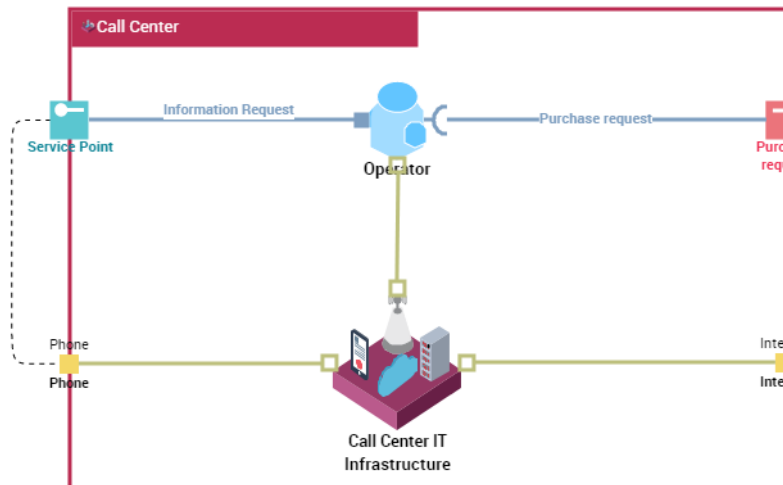
- For more details, see [Describing a Resource Architecture Environment](#).

Describing Resource Architectures

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

- For more details, see [Describing a Resource Architecture](#).

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.

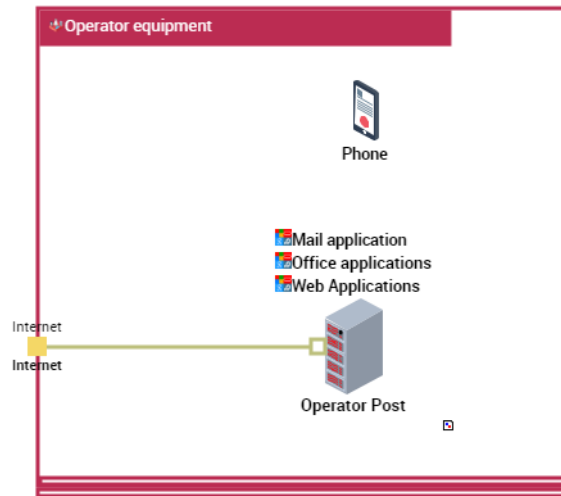
This diagram contains a *Request Point* from which the operators make purchasing requests.

) *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 an IT infrastructure](#).



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.

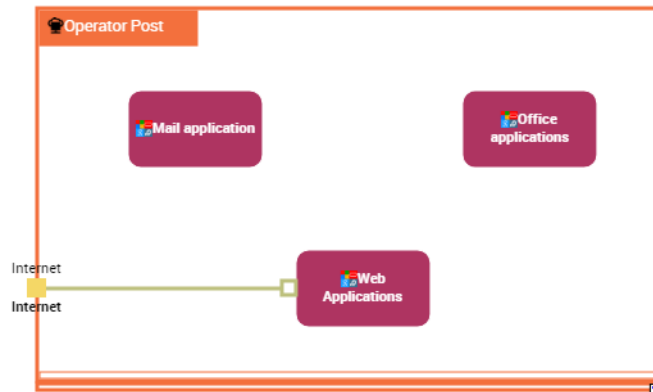
Note that the *Communication Protocols* used are specified on the communication channels.

) 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 application technical area and IT devices installed on a standard PC.

- For more details, see [Describing a Computing Device](#).



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.

Managing Service Catalogs

A service catalog describes the list of functionalities covered by a solution as well as the technical or functional elements that implement these functionalities.

-) A service catalog contains a list of key service offers for which solutions are recommended.

HOPEX IT Architecture V2 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.
- **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.

- **technical service catalogs,**
 -) *A technical service catalog provides a centralized information source for the technical services offered by the service provider organization. It contains a customer-oriented view of the technical services used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The technical service catalog presents the list of IT 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 V2 DESKTOP PRESENTATION

- **HOPEX IT Architecture V2** is mainly intended for web users. Desktops described in this guide are accessible only to Web desktop users.

If you have a **HOPEX IT Business Management** module, a specific desktop is available to you.

- For more details on the profiles and the facilities offered by the **HOPEX IT Business Management** module, see [Introduction to strategic transformation with HOPEX IT Business Management](#).

Connecting to the solution

To connect to **HOPEX IT Architecture V2**, see **HOPEX Common Features**, "HOPEX Web Front-End Desktop".

HOPEX IT Architecture V2 Desktop Presentation

The menus and commands available in **HOPEX IT Architecture V2** depend on the product licenses that you have and on the profile with which you are connected.

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

Presentation of space common to all profiles

All users, with the exception of users connected with the **Applications Contributor** and **Applications Viewer** profiles, have a **HOPEX IT Architecture V2** desktop and access to the following panes:

- **Home**: presents the main tiles useful for the user.
- **Dashboards**: displays the list of indicators required to steer objects such as processes, applications or org-units.
- **My Work**: displays the tasks assigned to the user.
- **Ideation**: used to manage the ideas, if you have the product license.
 - For more details ideas management, see «Submitting and evaluating ideas» chapter in the **HOPEX Common Features** guide .
 -
- **Reports**, enabling access to the group of reports available for 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 V2** reports, see [HOPEX IT Architecture V2 Reports](#).
- **Collaboration**, which enables access to all collaborative tools provided by **HOPEX**.
 - For more details on the use of collaborative tools, see "Accessing collaboration in **HOPEX**" chapter in the **HOPEX Common Features** guide .

HOPEX IT Architecture V2 Home space proposes a add tile button .

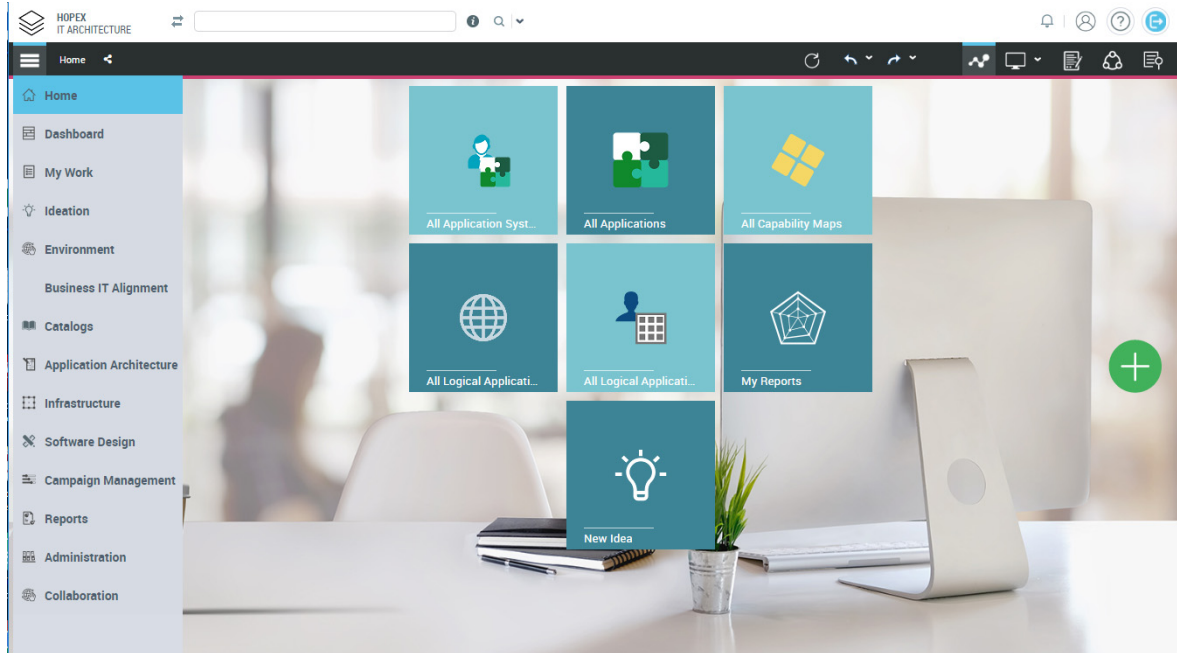
- See "Adding a tile to your home page" chapter in the **HOPEX Common Features** guide.

The panes provide access to the following menus:

- **History**, which contains all the objects you access or modify.
 - For more details on the use of history, see "Using the History" chapter in **HOPEX Common Features** guide.
- **Favorites**, to access to important objects and to usual actions.
 - For more details on the use of favorites, see "Managing Favorites" chapter in the **HOPEX Common Features** guide.

Presentation of the IT Architecture Functional Administrator space

In addition to the panes offered in standard mode to all **HOPEX IT Architecture V2** desktop users, users connected with the **IT Architecture Functional Administrator** profile have access to the panes described below:



The Environment pane

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

- **Organization**, to access the main objects processed with the **HOPEX IT Architecture V2** solution.
- **Standard Navigation**, to access the management features for libraries and Environments.
 - For more details, see [Preparing the Work Environment](#).
- **EA Projects**, to access the management features for projects.
 - For more information on managing projects, see "Enterprise Architecture (EA) projects in HOPEX in the **HOPEX Common Features** guide.

The Business IT Alignment pane

The **Business IT Alignment** pane provides access to the following menus:

- **Logical Architecture** , to describe the Logical Architecture elements.
 - *For more details on logical architectures, see [Modeling Business Capabilities and the logical application architecture](#).*
- **Portfolios**, to access the application portfolio management functionalities offered with the **HOPEX IT Portfolio Management** product.
 - *For more information, see **HOPEX IT Portfolio Management**.*
- **IT Architecture Projects**, to manage architecture projects.
 - *For more details on managing projects, see the "Projects in HOPEX" chapter in the **HOPEX Common Features** guide.*
- **Corrective Action Plans**, to describe and manage the action plans linked to the transformation of processes.
- **Controls & Risks**, for accessing the risk management features offered with the **HOPEX Risk Mapper** product.
 - *For more information, see **HOPEX Risk Mapper**.*

The Catalogs pane

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

- **Business Service Catalogs** to describe the user services offered by the information system.
- **Information Service Catalogs** to describe the user services offered by the information system.
- **Technical Service Catalogs** to describe the services offered by the technical elements of the information system.
- **Hardware Service Catalogs** to describe the services offered by the hardware elements of the information system.
- **Reports** to provide access to all the reports dedicated to the service catalogs.
 - *For more details on service catalogs, see [Using service catalogs](#).*

The Application Architecture pane

The **Application Architecture** pane provides access to the following menus:

- **Functional Architecture** to describe the functional elements of the information system.
 - For more details on functional and technical architectures, see [Modeling Technical and Functional Architectures](#).
- **Technical Architecture** to describe the technical elements of the information system.
 - For more details on technical elements, see [Describing IT Components](#).
- **Controls & Risks**, for accessing the risk management features offered with the **HOPEX Risk Mapper** product.
 - For more information, see **HOPEX Risk Mapper**.
- **Data Management** to access the management functionalities for business data offered with the **HOPEX Information Architecture** product.

The infrastructure pane

The **Infrastructure** pane provides access to the following menus.

- **Infrastructure** to describe the elements that make up the resource architecture.
 - For more details on infrastructures, see [Modeling IT Infrastructures](#).
- **Technical Architecture** to describe the applications and the technology stacks of the information system.
 - For more details on functional and technical architectures, see [Modeling Technical and Functional Architectures](#).

The software Design pane

The **Software Design (UML)** pane provides access to the following menus:

- **OO Implementation (UML)**, to model your IS with UML formalism.
 - For more details on the use of UML concepts, see [About UML implementation](#).

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

- To see those menus, open the **Options** window and check that **IT Architecture > HMI modeling and batch processing functionality (ADES)** option is activated.
- **Batch and Program Implementation**,
 - For more details, see [Describing Batch Processing](#).
- **User Interface Design**,
 - For more details on describing user interfaces, see the [Describing Batch Processing](#).

The Campaign Management pane

The **Campaign Management** pane provides access to the creation and management functionalities.

The Collaboration pane

The **Collaboration** pane makes available different means of communication in your **HOPEX IT Architecture V2** desktop.

- For more details on collaboration tools, see the "Accessing collaboration in **HOPEX**" chapter in the **HOPEX Common Features** guide.

The Administration pane

The **Administration** pane provides access to the user management features. The rights of different users on objects of imported libraries depend on their assigned profiles.

- For more details on the management users, see "Managing users" chapter in guide **HOPEX Common Features**.

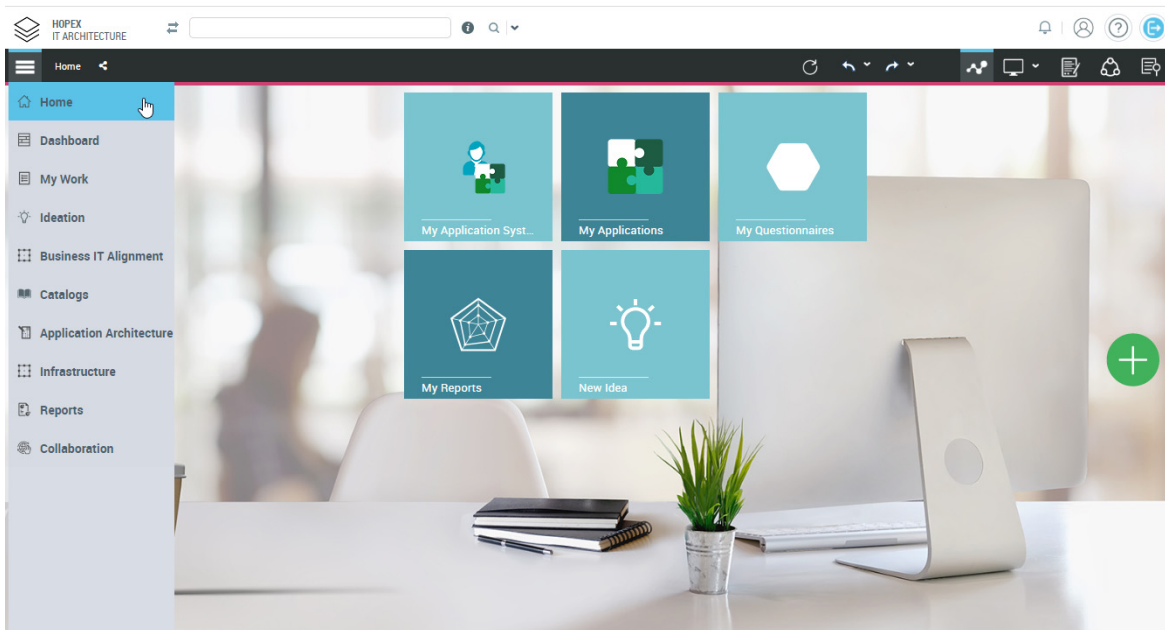
Presenting the IT Architecture Manager space

Users connected with the **IT Architecture Manager** profile have access to the same functionalities as users connected with the **IT Architecture Functional Administrator** profile. Only the administration activities are not offered.

- For more details, see [Presentation of the IT Architecture Functional Administrator space](#).

Presentation of the Application Architect space

In addition to the panes offered in standard mode to all **HOPEX IT Architecture V2** desktop users, the **Application Architect** has access to the **Business IT Alignment**, **Catalog** and **Application Architecture** panes.



The Business IT Alignment pane

The **Business IT Alignment** pane provides access to **Logical Architecture** menu, to describe the logical architecture elements.

- For more details on logical architectures, see [Modeling Business Capabilities and the logical application architecture](#).

The Application Architecture pane

The **Application Architecture** pane provides access to the **Functional Architecture** menu to describe the application systems of the information system.

- For more details on functional and technical architectures, see [Modeling Technical and Functional Architectures](#).

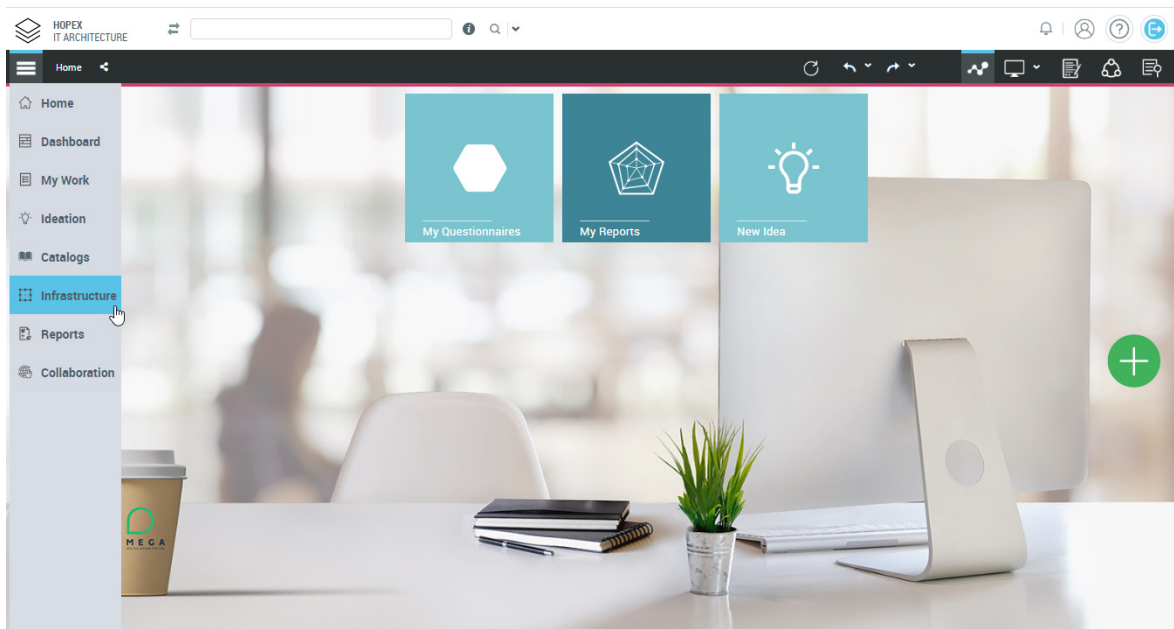
The Catalogs pane

The **Catalogs** pane provides access to the service catalogs.

- For more details on service catalogs, see [Using service catalogs](#).

Presentation of the Infrastructure Architect space

In addition to the panes offered in standard mode to all **HOPEX IT Architecture V2** desktop users, the **IT Infrastructure Architect** has access to the **Catalogs** and **Infrastructure** panes.



The Catalogs pane

The **Catalogs** pane provides access to the service catalogs.

- For more details on service catalogs, see [Using service catalogs](#).

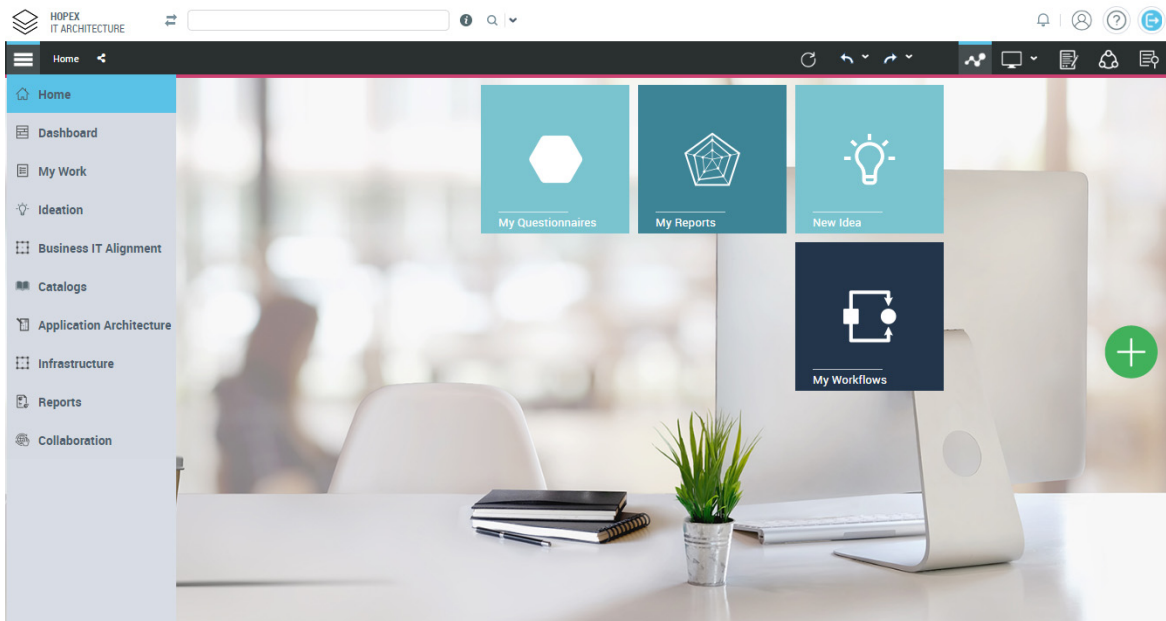
The infrastructure pane

The **Infrastructure** pane provides access to the following menus.

- **Infrastructure** to describe the elements that make up the resource architecture.
 - For more details on infrastructures, see [Modeling IT Infrastructures](#).
- **Technical Architecture** to describe the applications and the technology stacks of the information system.
 - For more details on functional and technical architectures, see [Modeling Technical and Functional Architectures](#).

Presentation of the IT Technical Architect space

In addition to the panes offered in standard mode to all **HOPEX IT Architecture V2** desktop users, the **IT Technical Architect** has access to the **Catalogs** and **Application Architecture** panes.



The Application Architecture pane

The **Application Architecture** pane provides access to the **Functional Architecture** menu to describe the application systems of the information system.

- For more details on functional and technical architectures, see [Modeling Technical and Functional Architectures](#).

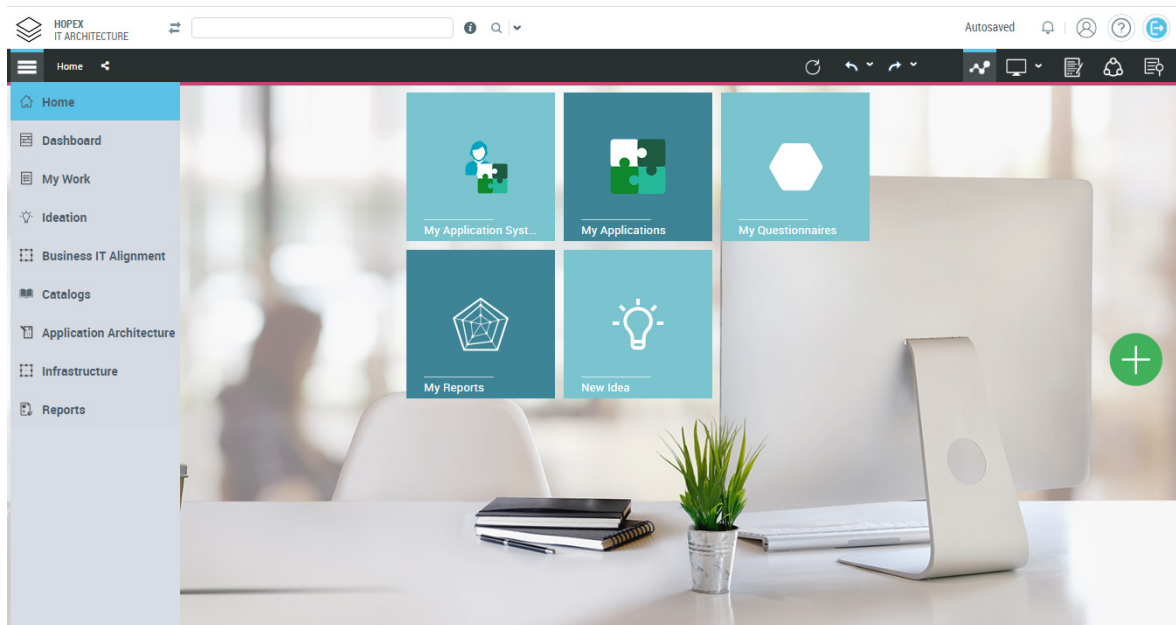
The Catalogs pane

The **Catalogs** pane provides access to the service catalogs.

- For more details on service catalogs, see [Using service catalogs](#).

Presentation of the Application Contributor space

Users connected with the **Applications Contributor** profile have access to the panes offered in standard mode to all users of the **HOPEX IT Architecture V2** desktop.

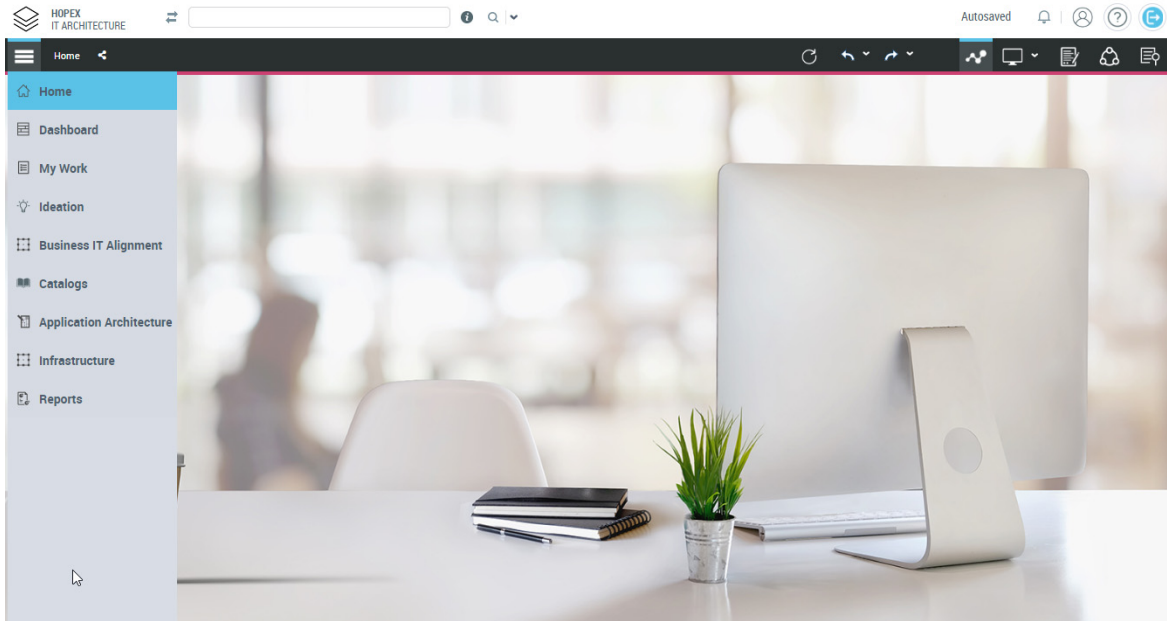


Users connected with the **Applications Contributor (lite)** profile have access to the **Enterprise Architecture** desktop for **HOPEX** solutions via the **HOPEX Explorer** application.

Presentation of the Application Viewer space

Users connected with the **Applications Viewer** profile have access to the panes offered in standard mode to all users of the **HOPEX IT Architecture V2** desktop.

Users connected with the **Application Viewer (Lite)** profile have access to the **Enterprise Architecture** desktop for **HOPEX** solutions via the **HOPEX Explorer** application.



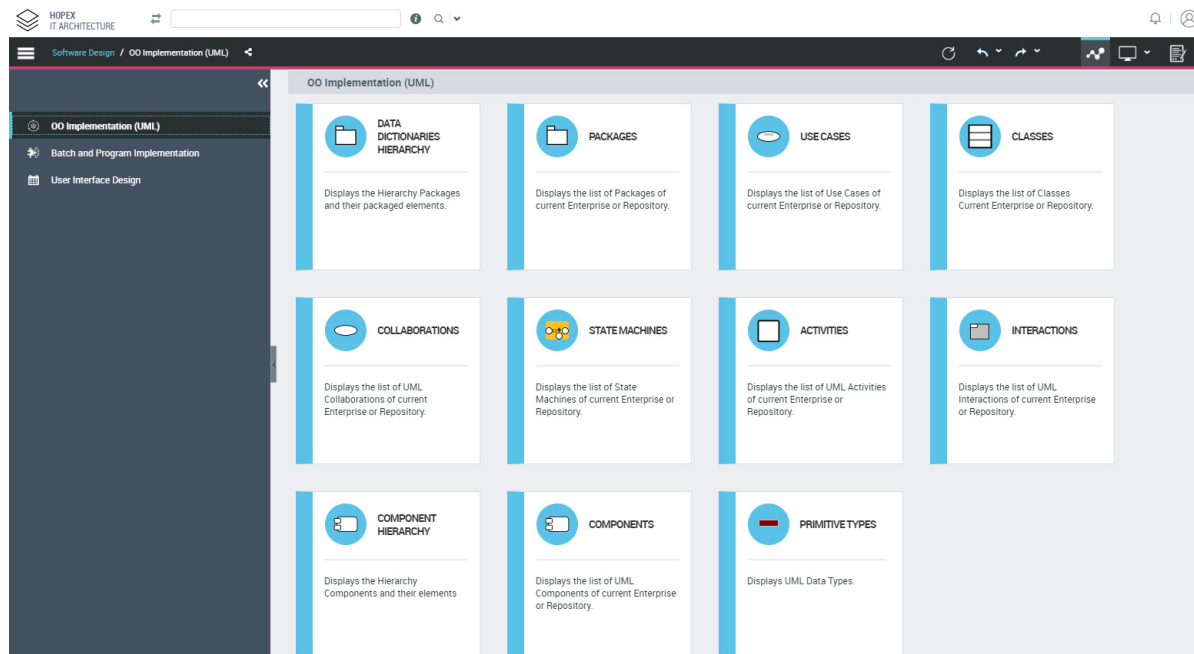
Presentation of the UML Designer desktop

In addition to the panes offered in standard mode to all users of the **HOPEX IT Architecture V2** desktop, the UML designer has access to the **Software Design** which provides access to the objects of the repository via the **OO Implementation (Us, sML)** navigation menu.

- For more details on the use of UML concepts, see [About UML implementation](#).

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

- To see those menus, open the **Options** window and check that **IT Architecture > HMI modeling and batch processing functionality (ADES)** option is activated.
- **Batch and Program Implementation,**
 - For more details, see [Describing Batch Processing](#).
- **User Interface Design,**
 - For more details on describing user interfaces, see the [Describing Batch Processing](#).



Switching between Profiles

Using the **HOPEX IT Architecture V2** 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 V2

Using conversion Tools to HOPEX V4

If you use **HOPEX V4** and if your administrator had carried out the conversion of **Business Person** instances coming from the previous repository into **Person (System)** instances, you must get 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 with **Easy Update** by your administrator.

The principle of this **HOPEX IT Architecture V2** 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 between 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	-	-	-	-

Asset Type	IT Manager	Process manager	Quality Manager	Risk Manager or Provider
Server	-	-	-	-
IT service	Local Application Owner	-	-	-
Workstation	-	-	-	-

Table of business roles used for created assignments

Preparing the Work Environment

In the context of the **HOPEX IT Architecture V2** 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.*

Accessing the list of libraries with HOPEX IT Architecture V2

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

- > Select **Standard Navigation > Standard Navigation**.
The library tree appears.

Creating a library with HOPEX IT Architecture V2

To create a library from the **Environment** navigation pane:

1. Select **Standard Navigation > Standard Navigation** in the navigation menu.
The library tree appears.
2. Right-click the **Library** folder and select **New > Library**.
A **Library** creation dialog box opens.
3. Specify the the name of the library.
4. If appropriate, enter the name of the **Owner**.
5. Click **OK**.
The library appears in the tree.

Using properties pages

Displaying the properties window on a permanent basis

You can choose to display the property windows in **HOPEX** on a permanent basis so as to view immediately the properties of an object.

To display the properties window on a permanent basis:

1. Click the **Properties**  button on the top right-hand side.
The **Properties** window appears in the Edit Area.
2. Select an object.
Its properties appear.

HOPEX IT Architecture V2 properties pages content

HOPEX IT Architecture V2 provides properties pages available for several solutions.

- 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 V2** objects.

- the **Components** pages provide access to the list of components of the described object defined in the different diagram types.
 - the **Components > Scenario** page provides access to the list of components of the described object defined in its flow scenarios.
 - For more information on a scenario of flow, see [Using a Scenario of Application Flows Diagram](#).
 - the **Components > Structure** page provides access to the list of components of the described object in its structure diagrams.
 - For more details on components of a structure diagram, see [Creating an Application Structure Diagram](#)
 - the **Components > Technical Architectures** page provides access to the list of components of the described object in its technical architecture diagrams.
 - For more information on the components of a technical architecture diagram, see [Creating an Application Technical Architecture Diagram](#).
- the **Use** pages provide access to the information relating to the use contexts of the the described object in different types of diagrams.
 - the **Usage > Scenario** page provides access to the list of elements that use the described object in their flow scenarios.
 - For more information on a scenario of flow, see [Using a Scenario of Application Flows Diagram](#).
 - the **Usage > Structure** page provides access to the list of elements that use the described object in their structure diagram.
 - For more information on the components of an application structure diagram, see [Creating an Application Structure Diagram](#).
 - the **Usage > Impact Report (Structure)** page is used to analyze the impact of the failure of the described object on the elements that use it or on the applications or application systems with which it interacts via exchange contracts as described in the structure diagrams that use the described object.
 - For more details, see [Impact report \(Structure\)](#).
 - the **Usage > Impact Report (Scenario)** page is used to analyze the impact of the failure of the described object on the application

systems, applications or application services with which it exchanges application flows in the context of the scenario of use of the described object.

- For more details, see [Impact report \(Scenario\)](#).
- The **KPI Dimension** page provides access to performance indicators connected to the described object.
 - For more details on managing key indicators, see the "Using KPIs" chapter in the **HOPEX IT Business Management** guide.
- the **Performed Process** page provides access to the application processes executed by the described object.
 - For more details on system processes, see [Describing Application Processes](#).
- the **Service and Request Points** page specifies the services expected or delivered by the described object.
 - For more details, see [Describing Service and Request Points](#).
- the **Assignment** page is used to specify the managers of the described object.
- The **Reports** page provides access to the reports available for the described object.
 - For more information on **HOPEX IT Architecture V2** reports, see [HOPEX IT Architecture V2 Reports](#).

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 **Environment** navigation pane:

1. Select **Organization > Org-Units**.
The list of all org-units appears.
2. Click the **New** button.
3. In the **Creation of Org-Unit** dialog box, enter the name of the org-unit you want to create.
4. 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. From the **Characteristics** tab, click in the **Internal/External** field and select "External" value.
3. Click on **OK** to apply the update and close the properties dialog box. This characteristic is represented graphically and is automatically displayed in the diagrams.

Importing components with HOPEX IT Architecture V2

HOPEX IT Architecture V2 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_Hardware_Functionalities_Import_Template.xlsx** for infrastructure elements import/export : computing devices connected to the breakdown of hardware functionalities they implement.
- **Technologies_Technical_Functionalities_Import_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 V2

HOPEX IT Architecture V2 Excel templates that enable import of hardware or technical elements are identically 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 combined 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 functionalities(technical or material), the elements are as follows:
 - **Technical or Hardware functionalities**,
 -) A functionality is a service required by an org-unit in order to perform its 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.
 - **Technical or Hardware functionality maps**,
 -) A functionality map is a set of functionalities with their dependencies that, jointly, define the scope of a hardware or software architecture.
 - For more details on functionalities, see [Using Functionalities with HOPEX IT Architecture V2](#).
 - **Sub-functionalities**, which define the link between a functionality and the functionality map (or the functionality) in which it is referenced.
 - **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 V2** is presented in the following order:

- For elements of type: *Hardware* or *technical functionality*, *Hardware* or *technical functionality 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 [Building the import file for HOPEX IT Architecture V2](#).

Several steps must be followed in order for the Excel import to be performed correctly:

1. [Checking the Excel import/export options](#),
2. (optional) [Specifying the current library](#),
3. [Importing objects in a repository](#),
4. [Importing objects in a repository](#).

- For more information on the structure of the Excel template to be imported, see [Building the import file for HOPEX IT Architecture V2](#).

Checking the Excel import/export options

To be able to use the Excel import/export features:

1. Open the options window, select folder **Data Exchange > Import/Export Synchronization > Tools/Third Party Formats**.
2. Check that the option **Export Excel: Availability in Listviews** is selected.

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.*

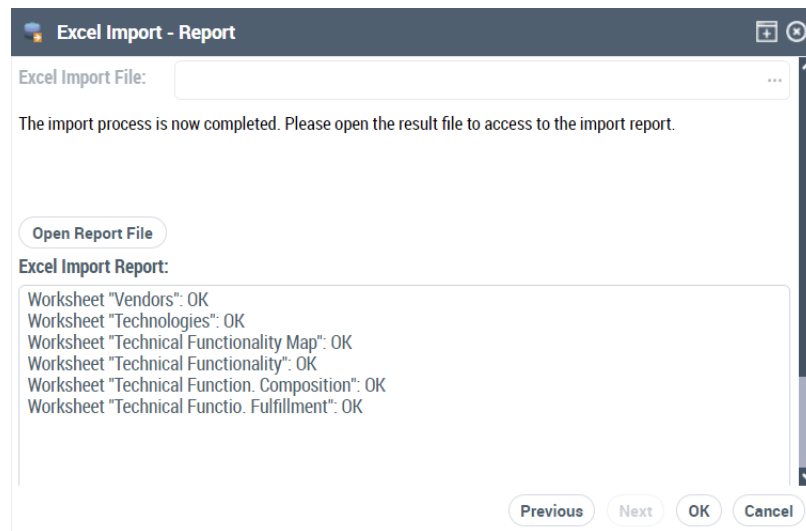
In order for the data you import with Excel to be linked to a specific container, you must specify the current library.

- For more details, see [Preparing the Work Environment](#).

Importing objects in a repository

To import objects using the Excel file of **HOPEX IT Architecture V2**:

1. Click the Main menu and select **Import > Excel (*.xls, *.xlsx)**. The import wizard appears in the edit window.
2. At the right of the **Excel Import File** field, click the **Browse** button.
3. Select the file to be imported.
 - For more information on the creation of the Excel file to be imported, see [Building the import file for HOPEX IT Architecture V2](#).
4. Click **Next**. The wizard provides a report of import results.



5. To obtain a detailed report of import errors, click the **Open Report** button. The .xls (or .xlsx) file opens indicating in color red the problem data.
6. To have the data imported into the current library, click **Import** or click **Import anyway**.
7. To modify the imported file or the import parameters, click **Previous**.
8. To discard import, click **Cancel**.

Building the import file for HOPEX IT Architecture V2

- For more information on the structure of the Excel template, see [Structure of the import/export Excel templates of HOPEX IT Architecture V2](#).

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 V2**.

- For more details on exporting data, see [Exporting components with HOPEX IT Architecture V2](#).

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 V2**.

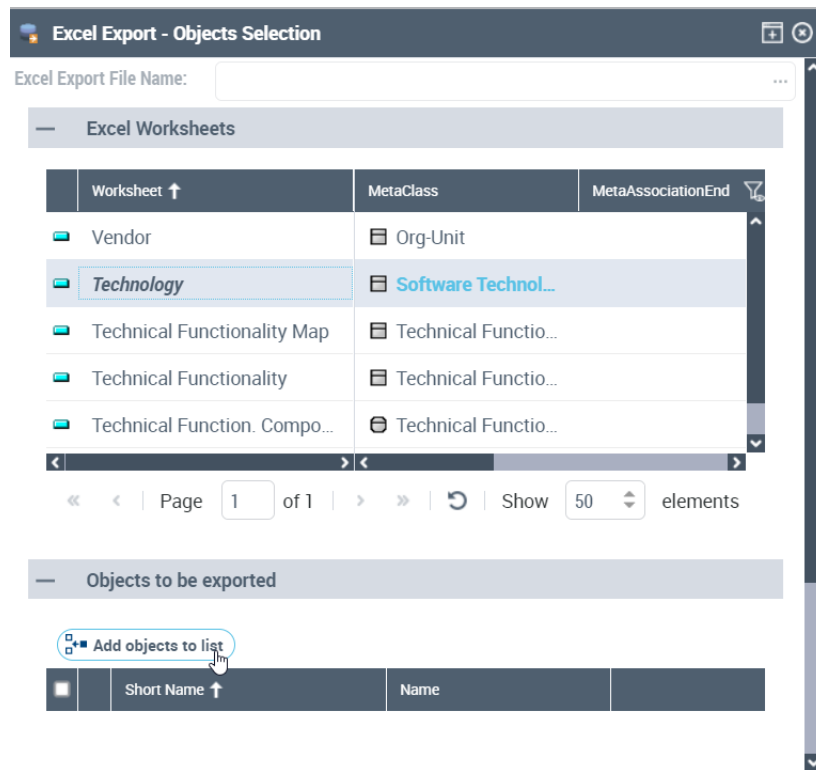
- For more details on additional information, see [Completing the import file for HOPEX IT Architecture V2](#).

Exporting components with HOPEX IT Architecture V2

To access the settings of the data export wizard from **HOPEX IT Architecture V2** to an Excel file:

1. Check that your export options are correct. See [Checking the Excel import/export options](#).
2. Click the Main menu and select **Import > Excel (*.xls, *.xlsx)**.
The export wizard appears in the edit window.
3. Select **From a template**.
4. In the filed **Predefined Template File** select **Technologies Functionality Template** if you want to import on the technology Excel template.
 - Select **Hardware Functionality Template** if you want to import on the hardware Excel template.
5. Click **Next**.
Export window appears to select the objects to be exported according to their type.

6. In the **Excel Worksheets** section, select the type of object you want to export and, in the **Objects to be exported** section, click **Add objects to list**.



7. From the query window, select the objects you wish to export.
8. When you have selected all the objects you want to export, click **Next**.
9. Click **Open the Excel file** to view the export file.
The file opens in an xlsx table. You can save it if you wish.
10. To modify export parameters, click **Previous**.
11. To discard export, click **Cancel**.
12. Click **OK** to finish.
The generated xlsx file is in the format expected for later import.

Completing the import file for HOPEX IT Architecture V2

For your import/export file to be correct, you must have specified the following elements:

- For each element of *Hardware* or *technical functionality*, *Hardware* or *technical functionality map*, *Vendor*, *Technology*, *Computing device* (It

devicee, *Server*, *IoT device*) type, you must enter the name of each object.

- For each breakdown (**Technical Function_Composition** 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.

P 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.

ABOUT THIS GUIDE

This guide explains how to make best use of **HOPEX IT Architecture V2** to ensure efficient management of IT Architecture.

Guide Structure

The first part of the **HOPEX IT Architecture V2** guide is composed of following chapters:

- **Modeling Business Capabilities and the logical application architecture** ; explains how HOPEX IT Architecture V2 helps you in analyzing your Logical Architecture.
- **Modeling Technical and Functional Architectures** ; presents the functionalities offered by HOPEX IT Architecture V2 to describe the IT components of your enterprise.
- **Modeling IT Infrastructures**; describes the functionalities proposed by **HOPEX IT Architecture V2** to take into account systems using resources other than software.
- **Using service catalogs**; explains how drawing up the list of elements that implement the functionalities of your SI.

The second part of the **HOPEX IT Architecture V2** 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 Assessment** guide, which describes functions proposed by **HOPEX** to use and customize assessment questionnaires.
- the **HOPEX Power Supervisor** administration guide.

Conventions used in the guide

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



Very important remark to avoid errors during an operation.

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

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



Architecture Specification



MODELING BUSINESS CAPABILITIES AND THE LOGICAL APPLICATION ARCHITECTURE



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:

- 6 Using Business Capabilities with HOPEX IT Architecture V2,
- 6 Using Functionalities with HOPEX IT Architecture V2,
- 6 Describing a Logical Application Architecture with HOPEX IT Architecture V2.

USING BUSINESS CAPABILITIES WITH HOPEX IT ARCHITECTURE V2

Business capabilities examples with HOPEX IT Architecture V2

A *business capability* defines an expected skill.

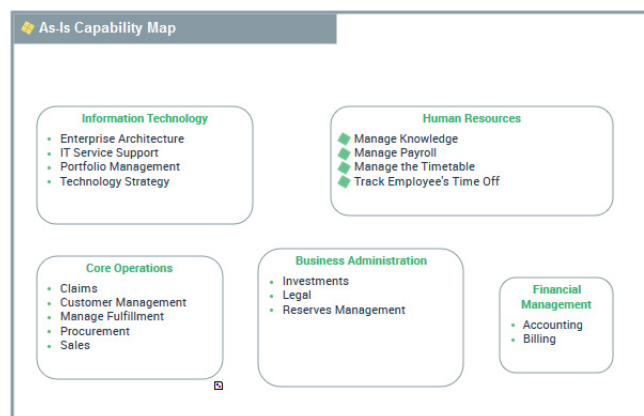
) .A business capability is a set of features that can be made available by a system (an enterprise or an automated system).

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 that, together, 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 Business**

Architecture & Strategic Planning , a report allows to check the functional coverage of your applications.

- For more details on the business capabilities reports, see [Business Capabilities reports](#).

Managing the Business Capability Maps with HOPEX IT Architecture V2

) A business capability map is a set of business capabilities with their dependencies that, together, define a framework for an enterprise stage.

Accessing the list of business capability maps

To access the list of capabilities maps from the **Business IT Alignment** navigation pane:

1. Select **Logical Architecture** in the navigation menu.
2. Click on **Business Capability Map** tile
The list of Capabilities maps appears in the edit area.

Creating a business capability map

To create a business capability map:

1. Select **Business IT Alignment > Logical Architecture** in the navigation menu.
2. Click on **Business Capability Map** tile
The list of Capabilities maps appears in the edit area.
3. Click the **New** button.
4. In the creation dialog box, enter the name of the business capabilities map and click **OK**.
The new business capability map is added to the list of existing capability maps.

Creating a business capability map diagram

To create a business capability map diagram:

1. Right-click the business capability map that interests you and select **New > Diagram**.
2. Select **Business Capability Map Diagram**.
The diagram opens in the edit area. The frame of the business capability map described appears in the diagram.
 - You can construct this diagram in tabular input mode.
 - Tabular input is available with the **HOPEX Web Front-End** module. For more details on the use of tabular input, see [Building a diagram in tabular entry](#).

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 V2**, 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 **Capability usage** page, which is used to identify the enterprises that use this 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](#).
- the **Assignment** page, which is used to specify the managers of the capability map.
 - For more details on other property pages proposed by **HOPEX IT Architecture V2**, see [HOPEX IT Architecture V2 properties pages content](#).

Managing Business Capabilities with HOPEX IT Architecture V2

) .A business capability is a set of features that can be made available by a system (an enterprise or an automated system).

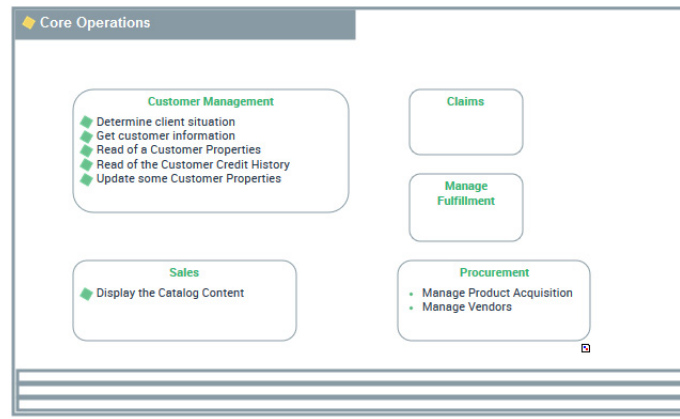
Describing a business capability

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".



Accessing the list of business capabilities with HOPEX IT Architecture V2

To access the list of capabilities from the **Business IT Alignment** navigation pane:

1. Select **Logical Architecture** in the navigation menu.
2. Click on **Business Capability** tile.
The list of business capabilities appears in the edit area.

Defining the functionalities associated with business capabilities

) A functionality is a service required by an org-unit in order to perform its 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.

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 V2](#).

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 V2](#).

USING FUNCTIONALITIES WITH HOPEX IT ARCHITECTURE

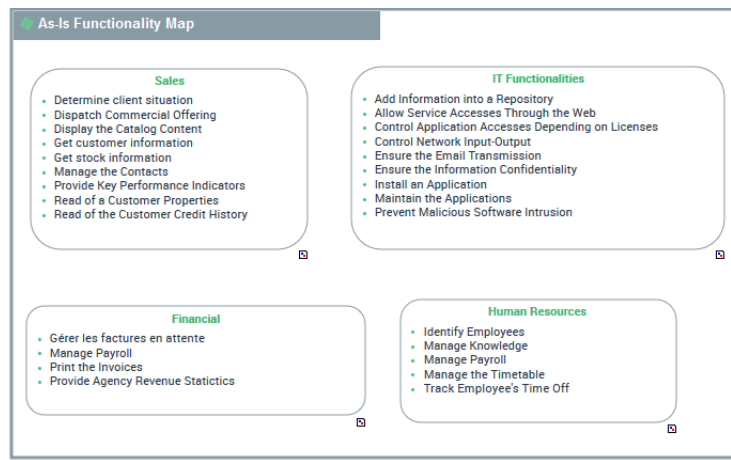
V2

A *functionality* is an aptitude expected from an equipment.

) A *functionality* is a service required by an org-unit in order to perform its 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 managing a functionality map, see the "Describing the Functionality Map" chapter in the **HOPEX IT Business Management** guide.

Describing a Functionality Map with HOPEX IT Architecture V2

) 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 V2

To access the list of functionality maps from the **Business IT Alignment** navigation pane:

1. Select **Logical Architecture** in the navigation menu.
2. Click on **Functionality Maps** tile.
The list of functionality maps appears in the edit area.

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 V2**, 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 [Creating a Functionality Diagram with HOPEX IT Architecture V2](#).*
- 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](#).*
- the **Capability usage** page, which is used to identify the enterprises that use this functionality map.
- the **Assignment** page, which is used to specify the managers of the functionality map.

Creating a functionality map

To create a functionality map diagram:

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

Describing functionalities with HOPEX IT Architecture V2

) *A functionality is a service required by an org-unit in order to perform its 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.

To access the list of Functionalities from the **Business IT Alignment** navigation pane:

- > Select **Logical Architecture > Functionalities** in the navigation menu.

The list of functionalities appears in the edit area.

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 capability effects, see [Creating a Functionality Diagram with HOPEX IT Architecture V2](#).

Creating a Functionality Diagram with HOPEX IT Architecture V2

To create a functionality diagram:

1. Right-click the functionality that interests you and select **New > 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 Technical Functionality

) A functionality is a service required by an org-unit in order to perform its 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.

With **HOPEX IT Architecture V2**, the use of technical functionalities and technical functional maps is identical to that of the functionalities and functionality maps.

– For more details on the operation of functionality maps and functionalities, see [Describing a Functionality Map with HOPEX IT Architecture V2](#) et [Describing functionalities with HOPEX IT Architecture V2](#).

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

1. Select **Technical Architecture** in the navigation menu.

2. Click on **Technical Functionality Maps** tile.
The list of technical functionality maps appears in the edit area.
 - To access the technical functionality list, select the **Technical Functionality** tab.

Describing a hardware functionality

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

With **HOPEX IT Architecture V2**, 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 V2](#) et [Describing functionalities with HOPEX IT Architecture V2](#).

To access the list of *hardware functionality maps* from the **Infrastructure** navigation pane:

1. Select **Infrastructure** in the navigation menu.
2. Click on **Hardware Functionality Maps** tile.
The list of hardware functionality maps appears in the edit area.
 - To access the hardware functionality list, select the **Hardware Functionality** tab.

DESCRIBING A LOGICAL APPLICATION ARCHITECTURE WITH HOPEX IT ARCHITECTURE V2

Describing a Logical Application System with HOPEX IT Architecture V2

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* is described by a structure diagram of the logical application system that represents the different components of the application systems and their interactions.

Accessing the list of logical application systems with HOPEX IT Architecture V2

To access the list of application systems from the **Business IT Alignment** navigation pane:

1. Select **Logical Architecture** in the navigation menu.
2. Click the **Logical Application Systems** tile.
The list of logical application systems appears.

Creating a Logical Application System

To create a *logical application system*:

1. From the **Business IT Alignment** navigation pane, select **Logical Architecture > Logical Application Systems**.
The list of application systems appears in the edit area.
2. Click **New**.
The **Creation of a Logical Application System** dialog box appears.
3. Enter the **Name** of your logical application system 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**.

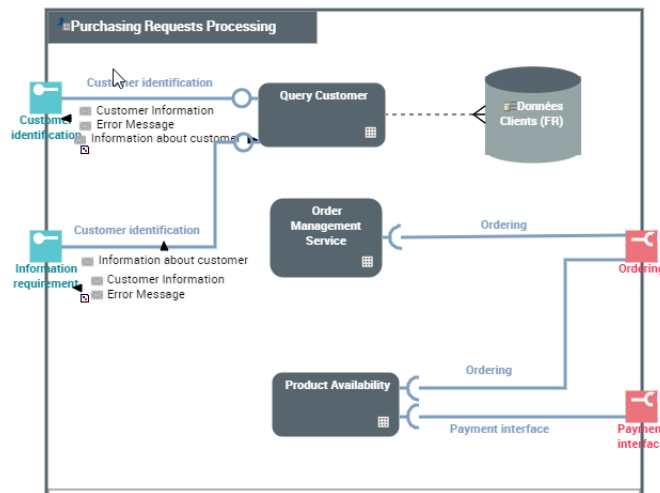
With **HOPEX IT Architecture V2**, a logical application system is described by the following pages:

- the **Properties** page, used to specify the properties that appear in the diagrams at the bottom of the described object frame.
- the **Structure** 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 with HOPEX IT Architecture V2](#).
- 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 V2**, see [HOPEX IT Architecture V2 properties pages content](#).

Describing a Logical Application System with HOPEX IT Architecture V2

With **HOPEX IT Architecture V2**, 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".



Structure diagram of the "Internet Purchase Requests" logical application system

- For more details on a logical application system representation, see [Describing a Logical Application System with HOPEX IT Architecture V2](#).

A structure diagram of the logical application system includes the following elements:

- **end users**
 -) The end user represents an organizational unit interacting at the boundaries of an application system or a logical application system.
 - For more details on adding end users, see [Adding an end user to the structure diagram of the logical application system](#).
- **Logical Application System Components and Logical Application Components**
 -) 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.
 - For more details on adding applications, see [Adding a logical application to the structure diagram of the logical application system](#).
- **interactions** between the components representing requests for services
 -) An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.
 - For more details on interactions between logical application system components, see [Managing Interactions](#).
- **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 information on access points, see [Describing Service and Request Points](#).

Adding an end user to the structure diagram of the logical application system

To create an **end user**:

1. In the objects toolbar of the structure diagram of the logical application system, 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.
 -) 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 the structure diagram of the logical application system

To describe that a logical application system implements a logical application:

1. In the objects toolbar of the structure diagram of the logical application system, 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 V2

) *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 V2

To access the list of logical application from the **Business IT Alignment** navigation pane:

1. Select **Logical Architecture** in the navigation menu.
2. Click the **Logical Applications** tile.
The list of logical applications appears in the edit area.

Creating a logical application

To create a *logical application*:

1. From the **Business IT Alignment** navigation pane, select **Logical Architecture > Logical Application Systems**.
The lists of logical applications appear in the edit area.
2. Click **New**.
The **Create a Logical Application** window appears.
3. 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**.
 - For more details on functionalities, see [Describing functionalities with HOPEX IT Architecture V2](#).

With **HOPEX IT Architecture V2**, 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 **Components, Executed Processes , KPI Dimension, Service and request points** as well as **Reports** pages are also available.
 - For more details on other property pages proposed by **HOPEX IT Architecture V2**, see [HOPEX IT Architecture V2 properties pages content](#).

Logical Application System Environment Description

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

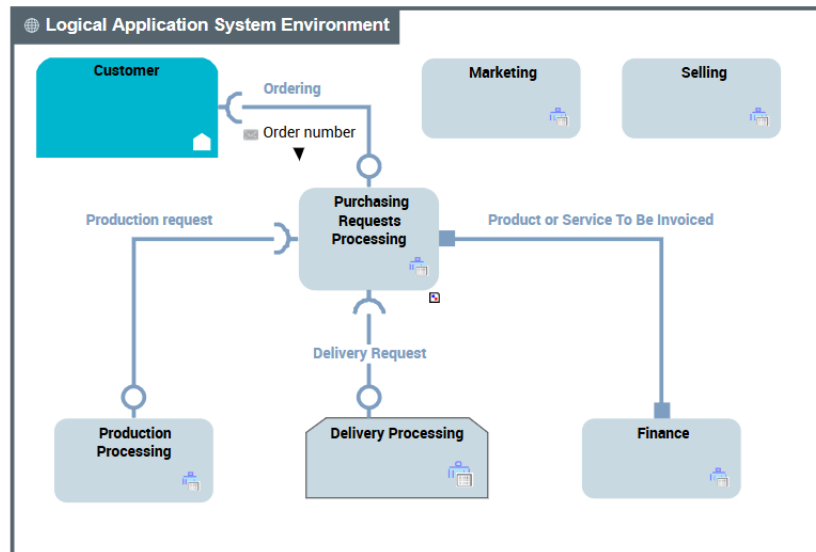
Example of logical application system environment

A logical application system diagram describes the 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 system environments from the **Business IT Alignment** navigation pane:

1. Select **Logical Architecture** in the navigation menu.
2. Click the **Logical Application System Environment** tile.
The list of logical application system environments appears in the edit area.

Creating a logical application system environment

To create a *logical application system environment*:

1. From the **Business IT Alignment** navigation pane, select **Logical Architecture > Logical Application System Environments**.
The list of logical application system environments appears in the edit area.

2. 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**.

With **HOPEX IT Architecture V2**, a logical application system environment 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.
 - For more details on other property pages proposed by **HOPEX IT Architecture V2**, see [HOPEX IT Architecture V2 properties pages content](#).

Using the Environment Structure Diagram of a Logical Application System

With **HOPEX IT Architecture V2**, a logical application system environment is described by a structure diagram of the logical application system environment that describes the interactions between the internal logical application systems, its users and the partner logical systems.

The 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 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.
- **Interactions** between the components representing requests for services.
) An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an

enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.

- For more details, see [Creating an Interaction](#).

USING IMPLEMENTATION MECHANISMS

This involves connecting the *business capability*, which corresponds to what we know how to do or what we want to do and which represents the goal (*the end*) to be achieved, to a way of achieving that which is represented by a *logical application* or a *logical application system* 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.*

Describing Fulfillment of a Business Capability

By 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 are made.*

Creating Fulfillment of a Business capability

A business capability can be achieved either by an application or application system, or at a conceptual level, by a logical application or 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 realization opens.
3. Select **Reusing an existing...**
4. Select the object type **Application** for example.
5. Select the application that interests you and click **OK**.
The capability realization appears in the list with the name of the selected application.

Analyzing enterprise capability implementation

HOPEX IT Architecture V2 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 [Breakdown map of business capabilities](#).

Describing the fulfillment of a Functionality

This involves connecting the *functionality*, which contributes to the goal to be achieved, to the means of implementation represented by *applications* (or *logical applications* at a conceptual level) or *application systems* (or *logical application systems*).

- Conceptual representations are made before organizational and technical choices are made.

Creating Fulfillment of a Functionality

A functionality can be achieved either by an application or application system, or at a conceptual level, by a logical application or application system.

To associate an application with a functionality, you must create a functionality realization.

-) 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 an application:

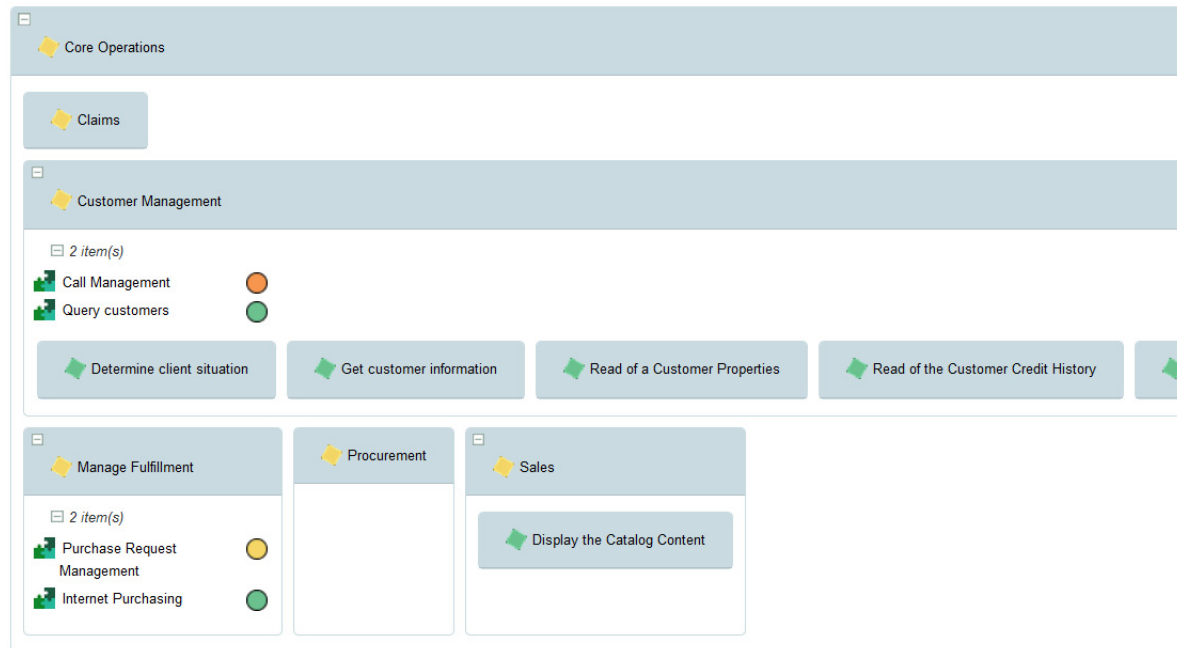
1. Open the **Fulfillments** property page of the functionality that interests you.
2. Click **New**.
The creation window for a functionality realization opens.
3. Select **Reusing an existing...**
4. Select the **Application** object type.
5. Select the application that interests you and click **OK**.
The functionality realization appears in the list with the name of the selected application.

- The components implemented by technical or hardware functionalities appear in the diagrams representing the functionality.

Identifying the logical applications associated with Functionalities

Applications cover *functionalities* associated with *business capabilities*. **HOPEX IT Architecture V2** 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 [Breakdown map of business capabilities](#).





MODELING TECHNICAL AND FUNCTIONAL ARCHITECTURES



HOPEX IT Architecture V2 is used by enterprises and other organizations to represent and document their 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 specific concepts that enable a more generic use of the tool.

The following points are covered here:

- 6 [Describing Application Architecture.](#)
- 6 [Describing Application Flows;](#)
- 6 [Managing Data;](#)
- 6 [Describing Application Processes.](#)
- 6 [Describing Technical Architecture.](#)
- 6 [Describing Batch Processing.](#)
- 6 [Defining User Interfaces.](#)

DESCRIBING APPLICATION ARCHITECTURE

HOPEX IT Architecture V2 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 V2](#).

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 an Application System](#).

Describing an Application with HOPEX IT Architecture V2

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.*

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

- An *scenario of application flows* describes the flows exchanged between the IT services or the micro-services used by this application. A scenarios can represent a particular application use case or more globally all the flows exchanged within this application.
 - For more details, see [Describing a scenario of flows](#).
- an *application structure diagram* is used to represent the interactions between the application components in the form of exchange contracts.
 - For more details, see [Creating a structure diagram](#).
- an *application technical architecture* used to represent the technical elements that support the application.
 - For more details, see [Creating an Application Technical Architecture](#).

Creating an Application with HOPEX IT Architecture V2

To create an *application*:

1. From the **Application Architecture** navigation pane, select **Functional Architecture > Applications**.
The list of applications appears in the edit area.
2. Select **My Applications** tab, for example.
3. Click **New**.
4. The **Creation of Application** dialog box appears.
5. Enter the **Name** of your application and click **OK**.
The new application appears in the list.

The properties of an application with HOPEX IT Architecture V2

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 **Type of application**
 - if it is an **Application template**: to be selected if the application is used to create other applications.
 - a **Comment**.
- the **business processes** that use the application.
 -) *A business process represents a system that offers products or services to an internal or external client of the company or organization. At the higher levels, a business process represents a structure and a categorization of the business. It can be broken down into other processes. The link with organizational processes will describe the real implementation of the business process in the organization. A business process can also be detailed by a functional view.*
- the **Functional Scope** of the application. See [Defining Application Functional Scope](#).
- the **Responsibility**: it relates to the person(s) responsible for the application.
 - Software Designer
 - Local Application Owner
 - IT Owner
 - *For more details on these roles, see [Business Roles of HOPEX IT Architecture V2](#).*
- the **Technologies** section provides access to the list of **Technologies** and the list of **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 a Software Technology](#).*
- **Exchanges** with other objects. See [Describing Application Flows](#).
- the **Risks** associated with the application. See [Specifying the Risks Associated with an Application](#).
- associated **Attachments**.
 - *For more details on other property pages proposed by **HOPEX IT Architecture V2**, see [HOPEX IT Architecture V2 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:
 - The *business capabilities* covered by the application
 -) *A business capability is a set of features that can be made available by a system (an enterprise or an automated system).*
 - For more details on business capabilities, see [Using Business Capabilities with HOPEX IT Architecture V2](#).
 - A report covers distribution of applications in business capabilities, see [Business Capabilities reports](#).
 - The *functionalities* implemented by the application
 -) *A functionality is a service required by an org-unit in order to perform its 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 V2](#).
 - The **Logical realizations** enable to define the *logical applications* and the *logical application systems* that the application implements.
 -) *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.*
 -) *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.*
 - For more details on logical architectures, see [Describing a Logical Application Architecture with HOPEX IT Architecture V2](#).

Specifying the Risks Associated with an Application

HOPEX IT Architecture V2 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 search window 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 an Application Environment with HOPEX IT Architecture V2

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

Accessing the List of Application Environments

To access the list of application environments from the **Application Architecture** navigation pane:

- > Select **Functional Architecture > Application Environments**.
The list of application environments appears in the edit area.

Creating an application environment

To create an *Application environment*:

1. From the **Application Architecture** navigation pane, select **Functional Architecture > Application Environments**.
The list of application environments appears in the edit area.
2. Click **New**.
The new application environment appears in the list.

Application environment properties

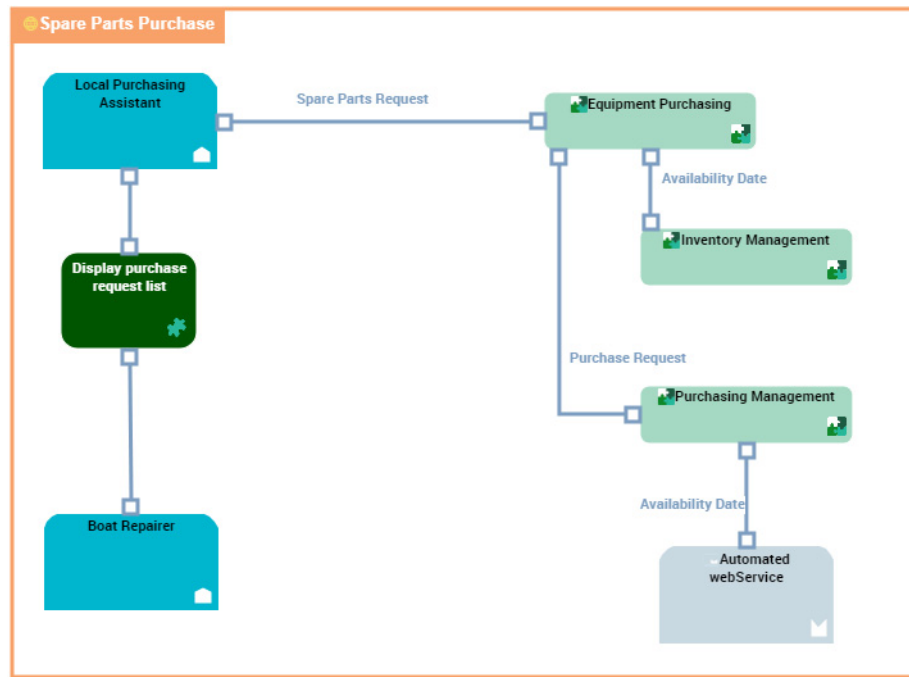
The **Characteristics** properties page of an application environment provides access to:

- 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 V2 properties pages content](#).*

With **HOPEX IT Architecture V2**, an *application environment* is entirely described by a an application environment diagram that is used to describe the interactions between the environment applications described, its users and the external applications.

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

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



"Spare Parts Purchase" application environment diagram

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

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

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

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

An application environment diagram includes:

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

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

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

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

This example concerns automated Web services.

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

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

This example concerns local participants.

) An org-unit represents 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.

- **interactions** between components.

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

- access, request and service points

Creating an application environment use case diagram

- For more details on use case diagrams, see [Use Case Diagram](#).
- To access these functions, you must be connected with **UML Designer**.

To create a use case diagram from an application environment:

1. Right-click the application environment and click **New > Diagram**.
2. Creating a **Application environment use case diagram**.
The diagram opens in the edit window.

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:

- a *scenario of application system flows* presents the flows exchanged between the application systems, the applications or the micro-services 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 [Describing a scenario of flows](#).
- an *application system structure diagram*, used to represent the interactions between the application components in the form of exchange contracts.
 - For more details, see [Creating an application system structure diagram](#).
- an *application system technical architecture* used to represent the technical architecture chosen for the deployment of each of the applications that support the application system.

Creating an Application System

To create an *application system*:

1. From the **Application Architecture** navigation pane, select **Functional Architecture > Application Systems**.
The list of application systems appears in the edit area.
2. Select **My Application Systems**.
3. Click **New**.
The **Creation of Application System** dialog box appears.
4. 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**,
 - a **Comment**.
- The **Business processes** section describes the list of business processes that use the application system.

) A business process represents a system that offers products or services to an internal or external client of the company or organization. At the higher levels, a business process represents a structure and a categorization of the business. It can be broken down into other processes. The link with organizational processes will describe the real

implementation of the business process in the organization. A business process can also be detailed by a functional view.

- About the **Functional Scope** section of the application system, see [Defining Application Functional Scope](#).
- 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 V2](#).*
- The **Attachments** section is limited to associated attachments.
 - *For more details on other property pages proposed by **HOPEX IT Architecture V2**, see [HOPEX IT Architecture V2 properties pages content](#).*

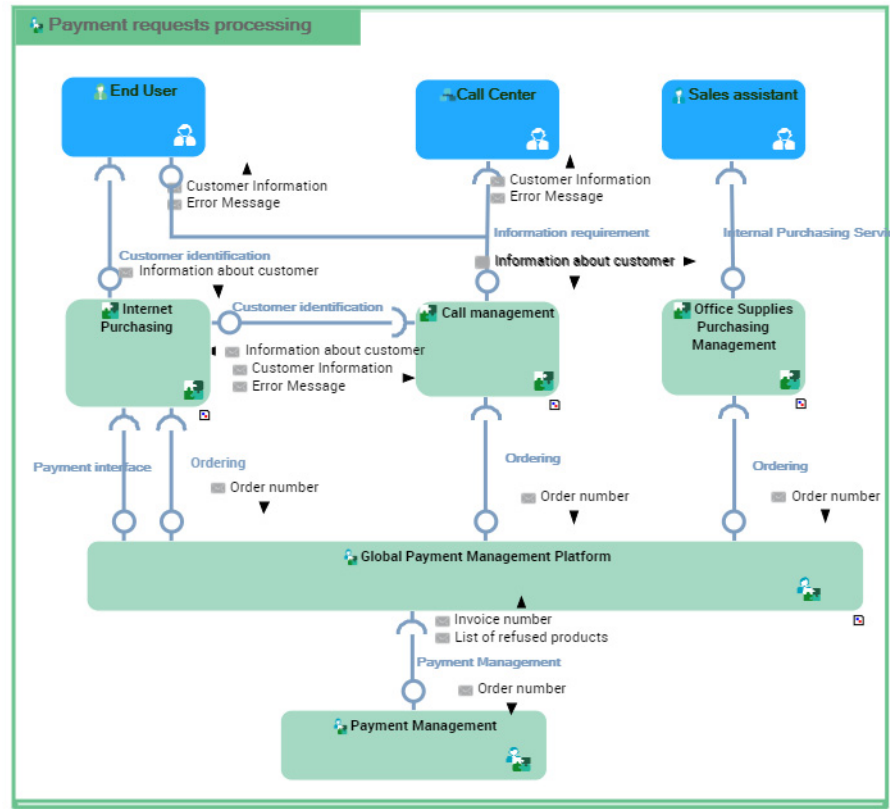
With **HOPEX IT Architecture V2** an application system is described by other property pages. See [HOPEX IT Architecture V2 properties pages content](#).

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, applications and micro-services,
- 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. Right-click the application system and select **New > Diagram**.
2. Select **Application System Structure Diagram**.

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 insert 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.

Describing an Application System Environment with HOPEX IT Architecture V2

) *An application system environment allows presenting the other application systems, applications or micro-services with which this application system can interact.*

Accessing the list of application system environments

To access the list of application system environments from the **Application Architecture** navigation pane:

- > Select **Functional Architecture > Application System Environment**.
The list of application system environments appears in the edit area.

Creating an application system environment

To create an *application system environment*:

1. From the **Application Architecture** navigation pane, select **Functional Architecture > Application System Environments**.
The lists of application system environments appear in the edit area.
2. Click **New**.
The **Creation of Application System Environment** window opens.
3. Enter the **Name** of your application system environment and click **OK**.
The new application system environment appears in the list.

Application system environment properties

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

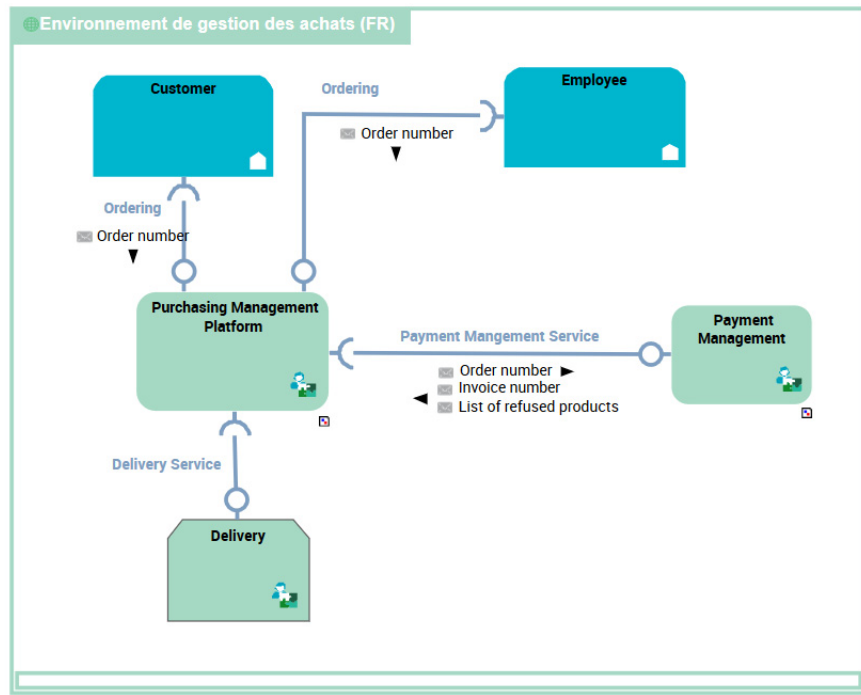
- its **Owner**, by default during creation of an application system environment, the current library.
- its **Name**,
- the text of its **Description**.

With **HOPEX IT Architecture V2** an application system environment is described by other property pages. See [HOPEX IT Architecture V2 properties pages content](#).

Describing an application system environment diagram

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

- For more details on use of a structure diagram, see [Creating a 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*.
 -) *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.*

This concerns two user categories: individuals and companies.

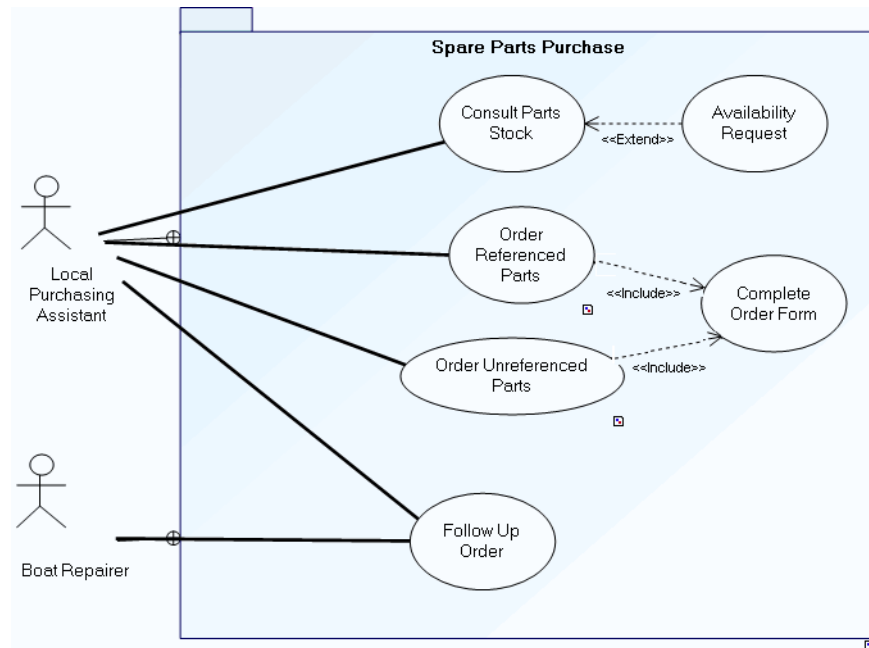
- *interactions* between components
 -) *An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units.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

Creating an Application Environment Use Case Diagram

By means of use case, **HOPEX IT Architecture V2** enables addition to description of an application architecture of the detailed specifications of its different sub-systems.

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

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



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

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

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

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

- For more details on use cases, see [Use Case Diagram](#).
- To access these functions, you must be connected with **UML Designer**.

DESCRIBING APPLICATION FLOWS

The Global Application Flow Map is the point of entry for modeling which is used to represent the main exchanges between the application systems and the applications.

Creating a Global Application Flow Map

An application flow global map is used to represent all the flows exchanged between the elements of the application architecture. The elements represented are:

- application systems,
- applications,
- architecture users.

The interactions offered between these elements are application flows that carry a content.

Accessing the list of global application flow maps

To access the list of global application flow maps using the **Application Architecture** navigation pane:

- > Select **Functionality Architecture > Global Application Flow Map**.
The list of global maps for application flows appears in the edit area.

The properties of a global application flow map

The complete description of a global application flow map is accessed from its properties pages.

The **Characteristics** properties page of the global application flow map provides access to:

- its **Owner**, by default, on creation of the global application flow map, the current library.
- its **Name**,
- the text of its **Description**.

The **Scenario** property page of a global application flow map provides access to the list of its components: application systems, applications, participants and flows.

Using a Scenario of Application System Flows

– For more details on the use of a scenario of flows, see [Describing a scenario of flows](#).

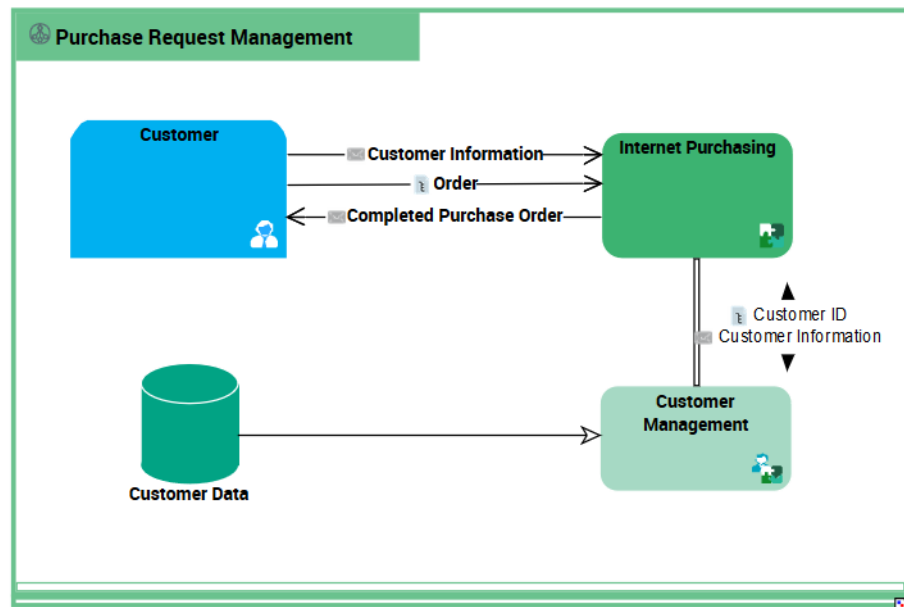
A scenario of application system flows represents the flows exchanged between certain elements of the application system in a given context. The elements represented are:

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

The interactions offered between these elements:

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

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:

- > Right-click the application system and select **New > Scenario of Application System Flows**.

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

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 [Reinitialize components in a scenario of flows](#).

Describing a Scenario of Application System Environment

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

- For more details on use of a scenario of flows, see [Describing a scenario of flows](#)

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

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* references an *data domain*.

) *A data area represents a restricted data structure dedicated to the description of a software Data Store. It is made of classes and/or data views and can be described in a Data Area Diagram.*

Using Data Stores

Introduction to data store concept

) *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 chapter "Logical and application data areas" in the **HOPEX Information Architecture** guide.*

Accessing to data areas with HOPEX IT Architecture V2

To access the list of data areas from the **Application Architecture** navigation pane:

- > Select **Data management**.
Two tiles enable you to access to the **Logical data areas** and to the **Application data areas**.

Creating a data area

To create an application data area, for example:

1. From the **Application Architecture** navigation pane, select **Data management > Application data areas**.
The lists of application data areas appear in the edit area.
2. Click **New**.
The new application data area appears in the list.
3. Modify the **Name** of your application data area.

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

A data store references an *data domain*.

) *A data area represents a restricted data structure dedicated to the description of a software Data Store. It is made of classes and/or data views and can be described in a Data Area Diagram.*

- *For more information on data area, see chapter "Logical and application data areas" in the **HOPEX Information Architecture** guide.*

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

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

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

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

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

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

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

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

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

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

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

Usage contexts

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

Data store type	Diagrams
Logical data store	Logical application system structure diagrams
Physical data store	Structure diagrams - of application, - of application system, - IT service, - of micro-service.
Application data stores	Scenario sequence diagrams - of application, - of application system, - IT service, - of micro-service. Scenario of flows diagrams - of application, - of application system, - IT service, - of micro-service.

Creating a local data store

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

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

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

Creating an external data store

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

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

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

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

Describing access to a data store

To create a read access to the data store:

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

UML modeling of data

HOPEX IT Architecture V2 provides the tools required to model logical data via class diagrams and data models.

- *To access these functions, you must be connected with **UML Designer**.*

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 Information Architecture** guide.*

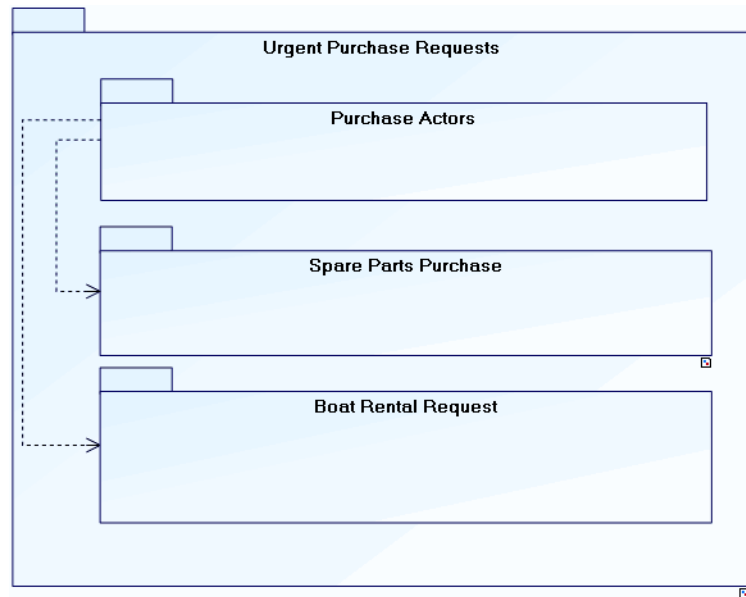
The different logical view concepts are described in the paragraph.

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

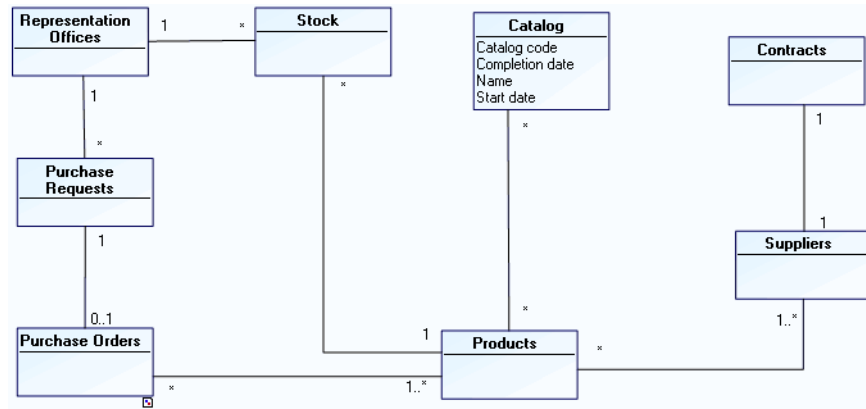
Like a package, a data model is used to represent the static structure of a system, particularly the types of objects handled in the system, their internal structure, and the relationships between them.

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

For more details on creating and updating a data model, see ["Le modèle de données", page 9](#).

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 [Using Data Stores](#)). 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.

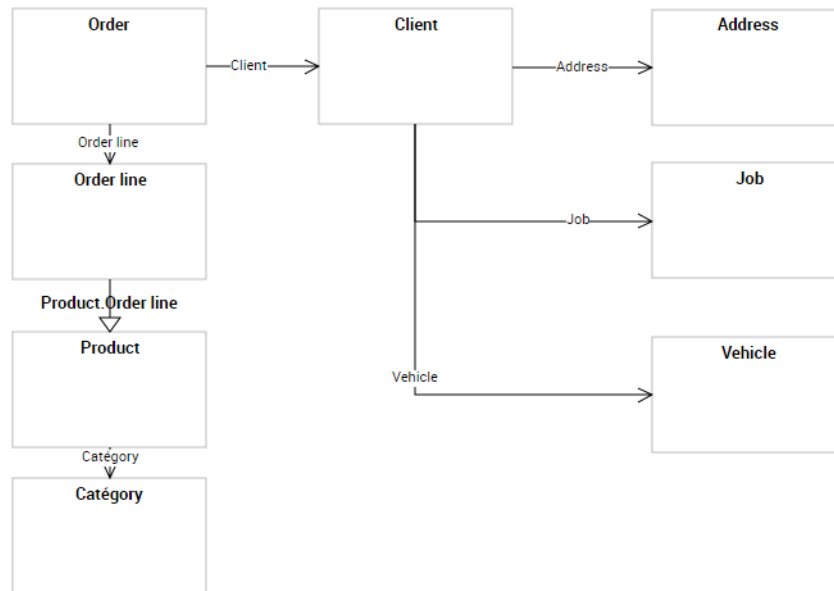
A logical data domain is owned by a package and can reference objects held in other packages.

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.

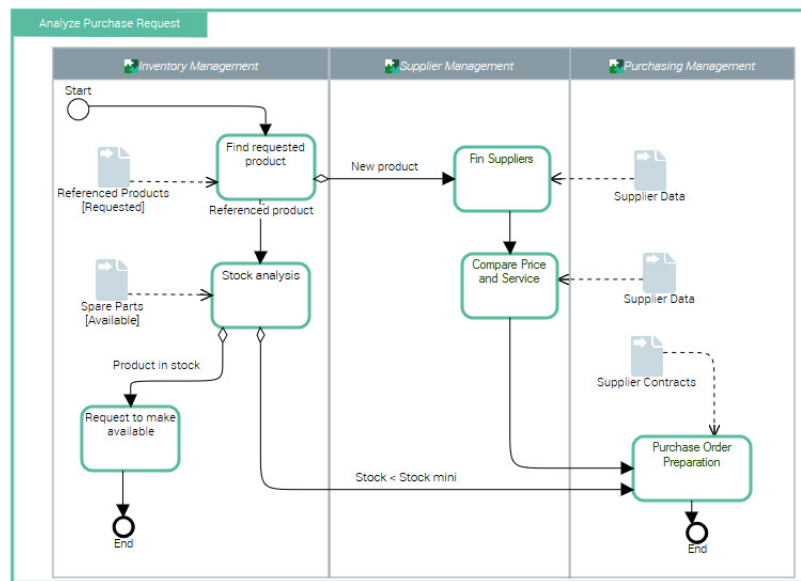
DESCRIBING APPLICATION PROCESSES

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

System process 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 V2

) 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.

Accessing system processes

To access the list of application processes from the **Application Architecture** navigation pane:

1. Select **Functional Architecture**.
2. Select **System Processes > All System Processes** tile.
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 "Entering a diagram in tabular 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 entry mode selection dialog box opens.
2. Click **No** to open the diagram editor.
The diagram opens in the edit area. You are now in the **HOPEX** graphic editor. The frame of the system process described appears in the diagram.

– For more information on initialization of a process diagram, see the chapter "Initialization of a BPMN diagram" in the **HOPEX Business Process Analysis** guide.

To open a system process diagram:

- > Right-click the process name and select **System Process 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.

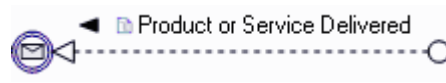
To create a task:

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.

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

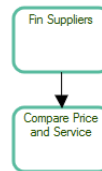


- A message flow can be linked to an event of message type.

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 exchange 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

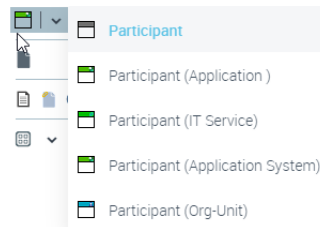
-) Gateways are modeling elements that are used to control how sequence flows interact as they converge and diverge within a process.
- For more information on managing gateways, see "Using gateways" chapter in guide HOPEX Business Process Analysis.

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

Complying with BPMN standard, a process can have different behaviors. With **HOPEX Business Process Analysis**, these behaviors are available for organizational processes, operations, system processes and tasks.

Behaviors

Behaviors proposed are:

- **Transaction**: a transaction is a set of coordinated activities leading to a consistent, and verifiable outcome.
- **Loop**: a loop is a process step that is repeated as long as a condition is true.
 - "Do while": the condition is evaluated before the first execution.
 - "Do until": the condition is evaluated after the first execution. In this case, the process step is executed at least once.The predicate enables specification of the loop execution condition.
- **Ad hoc**: steps of an ad hoc process are not controlled or sequenced in a particular order. Their performance is determined by the performers of the process.
- **Multiple**: the process is repeated a predefined number of times, evaluated only once before it is carried out. Execution type can be specified:
 - "Parallel": all executions carried out simultaneously.
 - "Sequential": executions carried out one after the other.
- **Compensation**: a compensation defines the set of activities that are performed during the roll-back of a transaction to compensate for activities that were performed during the normal flow of the process.

To describe for example that a system process is executed by a loop:

1. Open the **Characteristics** property page of the process.
2. In the **Details** section, in the **Loop** field, select the loop type corresponding to the process behavior and add the condition text. Shape of the process is modified to display the symbol of the loop.



Task type

To specify the type of a task:

1. Open the **Characteristics** property page of the process.

2. In the **Details** section, click the arrow at the right of the **Task Type** box. A list of task types appears.
 - **Call Process**: task used to call a second process while executing the current process.
 - **Receive**: elementary task which waits for arrival of a message from a participant external to the process. When the message has been received, the task is completed.
 - **Send**: task that sends a message to a participant external to the process. When the message has been sent, the task is completed.
 - **Manual**: task executed without the help of a automatic execution engine of a process or IT application.
 - **Business Rule**: execution task of a business rule with a rules engine which processes input data and returns calculation results.
 - **Script**: task executed by a process execution engine. The designer defines a script in a language that the engine is able to interpret. When the task is ready to start, the engine executes the script. The task is completed when script execution is completed.
 - *Shape of the process is modified to display the symbol associated with the task type.*

Describing the 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 .

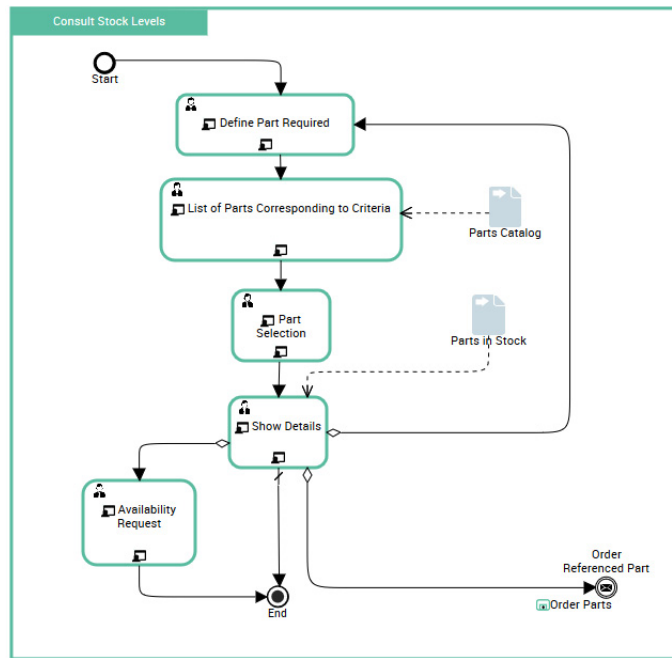
) *A system process is the executable representation of a process. the events of the workflow, the tasks to be carried out during the processing, the algorithmic elements used to specify the way in which the tasks follow each other, the information flows exchanged with the participants.*

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

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

1. Right-click the system process, for example "Consult Stock Levels" to open its pop-up menu.

2. Select **System Process Diagram**.
The diagram associated with the process opens.

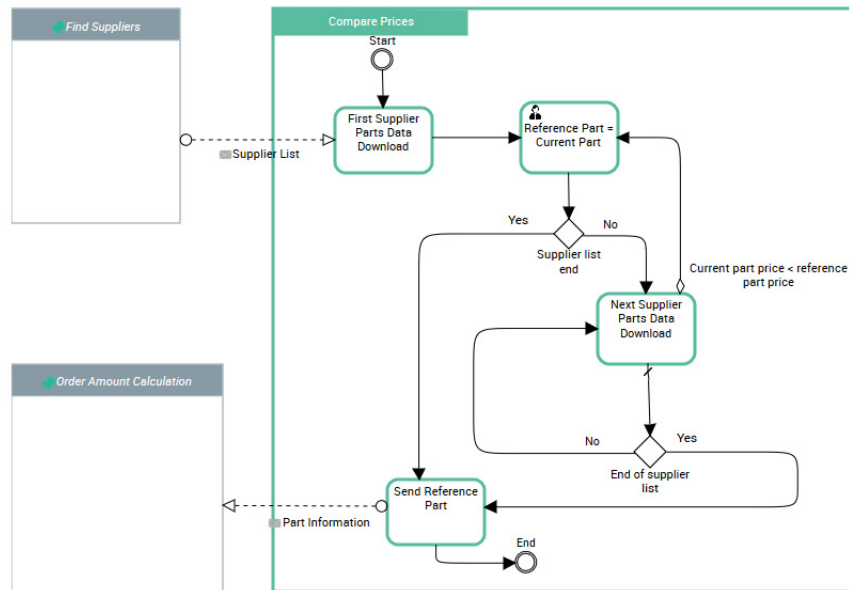


Consulting stock levels begins by display of a screen enabling identification of the required part. The list of parts found in the catalog is presented in the next screen. When the user has selected the required part, information on details is displayed. From this screen, it is possible to obtain information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

Modeling Tasks of an IT Service

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

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



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

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

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

DESCRIBING TECHNICAL ARCHITECTURE

A technical architecture allows you to represent the elements that must be deployed to implement an application: *Application Technical Areas*, *Technical Data Areas* as well as *Technical Communication Lines* used for data exchange.

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 **Application Architecture** navigation pane:

- > Select **Technical Architecture > 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 technologie, 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 technical functionalities covered by this software technology.

– *For more details on functionalities, see [Describing functionalities with HOPEX IT Architecture V2](#).*

Describing Software Technology Stacks

-) 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 **Application Architecture** navigation pane:

- > Select **Technical Architecture > Software Technology Stacks**.
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:

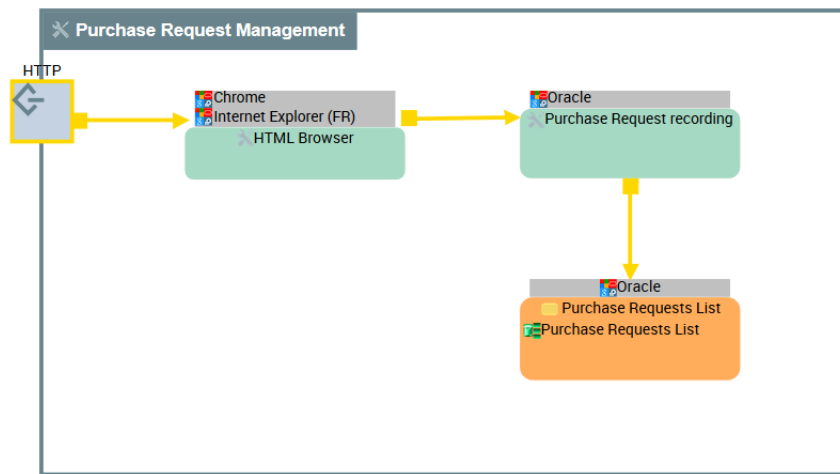
- **Components** which provides access to the list of concerned software technologies,
- **Responsibility**,
- **Owned Realizations** which represent the list of technical functionalities covered by this software technology stack.
 - For more details on functionalities, see [Describing functionalities with HOPEX IT Architecture V2](#).

Creating an Application Technical Architecture

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

The application technical architecture below presents the technical application areas used by the "Purchase Request Management" application. The technical communication line

between the communication port and the "HTML Browser" is based on the HTTP communication protocol.



Example of the application technical architecture for "Purchase Request Management".

An application technical architecture is described by an application technical architecture diagram composed of the following elements:

- **Application Technical Areas,**
 -) An application technical area represents an organizational element of an application according to technical criteria. For example, this can be the user interface or a process. Each application technical area is associated with one or more technologies. The deployment of several application technical areas is necessary for the application to be operation.
- **Technical Data Areas,**
 -) A data technical area represents an organizational element of an application used to access the data necessary for the operation of this application. Each application technical area is associated with one or more technologies (E.g.: Oracle 12, SQL Server 2012, etc.). A data technical area can allow access to one or more data stores.
- **Technical Server Port and Technical Client Port,**
 -) A technical server port is a point used to open communications with a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).
 -) A technical client port is a point used to request the opening of communications from a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).
- **Technical Communication Lines.**
 -) A technical communication line represents a technical connection between architectures or application technical areas through client and server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.
 - For more details on technical architecture diagram, see [Creating an Application Technical Architecture Diagram](#).

Accessing the application technical architectures

To access the list of application technical architectures from the **Application Architecture** navigation pane:

- > Select **Technical Architecture > Technical Architectures**.
The list of technical architectures appears in the edit area.

Properties of an application technical architecture

The complete description of an application technical architecture can be accessed from its property pages.

The **Characteristics** property page of an application technical 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 V2** an application technical architecture is described by other property pages.

- The **Component** property page which provides access to the list of components of the described technical architecture.
 - List of **Technical areas**: *technical application areas* and *technical data areas*,
 - List of **Technical ports**: *Technical Server Port* and *Technical Client Port*.
 - For more information on the components of a technical architecture diagram, see [Creating an Application Technical Architecture Diagram](#).
- the **Usage** property page provides access to the information relating to the use contexts of the described technical architecture.
- **Reports** page, used to access the different reports available on the described application technical architecture.

Creating an Application Technical Architecture Diagram

To create an Application Technical Architecture Diagram:

1. Right-click the Application Technical Architecture that interests you and select **New > Diagram**.
2. In the dialog box, select **Application Architecture Structure Diagram**.

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

Adding an application technical area to an application technical architecture diagram

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

To create an **application technical area**:

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

You can add data technical areas in the same way.

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

Defining the software technologies used by an application technical area

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

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

1. Open the **Characteristics** property page of the **Application Technical Area** that interests you.
2. In the **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 for the application technical area.

Technical ports

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

) A technical server port is a point used to open communications with a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).

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

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 in the described technical architecture frame.
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 technical communications

The communication between the application technical areas and the technical data areas can be described by technical communication lines. A technical communication line supports the *network application protocol* defined to create the communication.

) A technical communication line represents a technical connection between architectures or application technical areas through client and server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.

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

To create a *technical communication line*:

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

Describe an application system technical architecture

) An application system technical architecture describes one of the configurations possible for deploying the applications of a system application. It describes which technical architecture is chosen for each of the applications that it contains and the communication protocols that they use to communicate with each other.

An application system technical architecture is described by a technical architecture diagram composed of the following elements:

- *Application Technical Architectures*,
) An application technical architecture describes one of the configurations possible for application deployment. It describes how the different technical areas of the application are connected to each other and the technologies and the communication protocols that they use. An application can have a number of possible technical architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)
- *Partner micro-services*,
) A micro-service is a software component that can be deployed autonomously, but which does not directly provide an end user service. It can interact with other application services, applications or application systems. This is a deployable software component that uses software

technologies. For example: an authentication service, a PDF file printing service.

- **Technical Data Areas,**
 -) A data technical area represents an organizational element of an application used to access the data necessary for the operation of this application. Each application technical area is associated with one or more technologies (E.g.: Oracle 12, SQL Server 2012, etc.). A data technical area can allow access to one or more data stores.
- **Technical communication lines.**
 -) A technical communication line represents a technical connection between architectures or application technical areas through client and server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.
 - For more details on technical architecture diagram, see [Creating an Application Technical Architecture Diagram](#).

Accessing the list of application system technical architectures

To access the list of application system technical architectures from the **Application Architecture** navigation pane:

- > Select **Technical Architecture > Application System Technical Architectures**.

The list of application system technical architectures appears in the edit area.

Properties of an application system technical architecture

The complete description of an application system technical architecture can be accessed from its property pages.

The **Characteristics** property page of an application system technical 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 V2** an application system technical architecture is described by other property pages.

- The **Technical Architecture** property page which provides access to the list of components of the described technical architecture.
 - List of **Deployment Architectures**:
 - List of **Data Components**:
 - List of **Flows**.
 - For more information on the components of an application system technical architecture diagram, see [Creating an Application Technical Architecture Diagram](#).
- **Reports** page, used to access the different reports available on the described application technical architecture.

Creating an Application System Technical Architecture Diagram

The application system technical architecture description procedure is identical to that for application technical architecture. For more details, see [Creating an Application Technical Architecture Diagram](#).

DESCRIBING BATCH PROCESSING

With **HOPEX IT Architecture V2**, you can describe the sequencing of automated processing in **batch planning structure diagrams**.

- To access these functions, you must be connected with **UML Designer**.
- 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 more 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**.

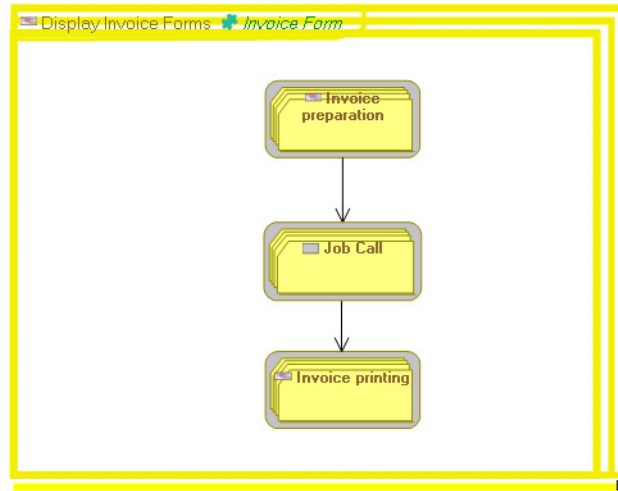
- For more details, see [Creating a Batch Program Structure Diagram](#).

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 the **Software Design** navigation pane.
2. In the navigation pane, select **Batch and Program Implementation**.
3. Click on **Batch and Program** tile.
The list of batch plannings appears.
4. Right-click the batch planning that interests you and select **New > Batch Planning Structure Diagram**.
The diagram opens.

Adding a call for batch processing in the diagram

The components of a **batch 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:

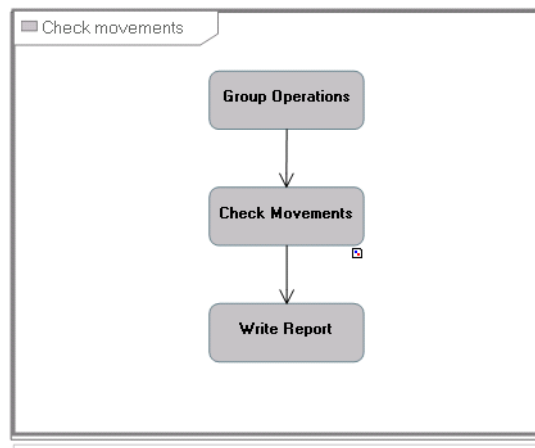
1. Click **Batch Sequence**.
2. Click the initial batch processing call and, holding the left mouse button down, draw a link to the batch processing call.
3. Release the mouse button.
The link representing the sequencing of the processes appears in the diagram.

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

Adding a programming call to the diagram

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

To add a component to a diagram:

1. Select **Programming Call** and click in the diagram.
The **Add a Programming Call** dialog box opens.
2. Click the arrow at the right of the **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 window for creating a system process batch realization appears.
3. Select the application process that interests you and click **Add**.
4. Click **OK**.
The system process batch realization appears in the properties page of the batch plan.

DEFINING USER INTERFACES

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

- To access these functions, you must be connected with **UML Designer**.
- 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 interface :

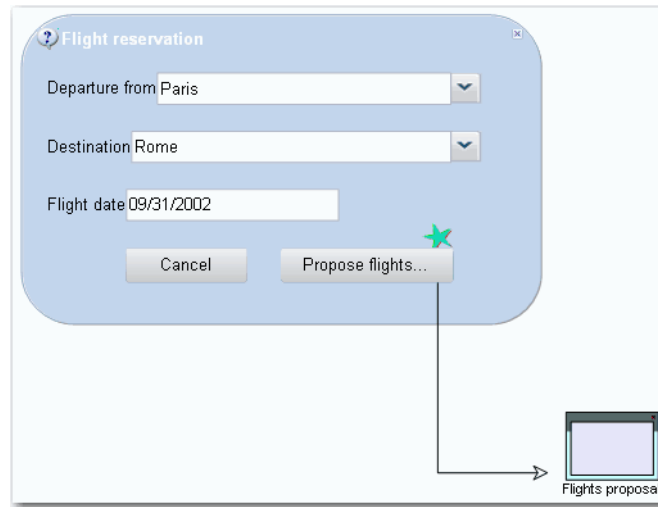
1. Click the **Software Design** navigation pane.
2. In the navigation pane click **User Interface Design**.
The list of user interfaces appears.
3. Click **New**.
4. Enter the name of the interface.
5. Click **OK**.

Building a User Interface Diagram

To create an interface diagram:

1. Right-click the user interface in question.
2. Select **New > Diagram > 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

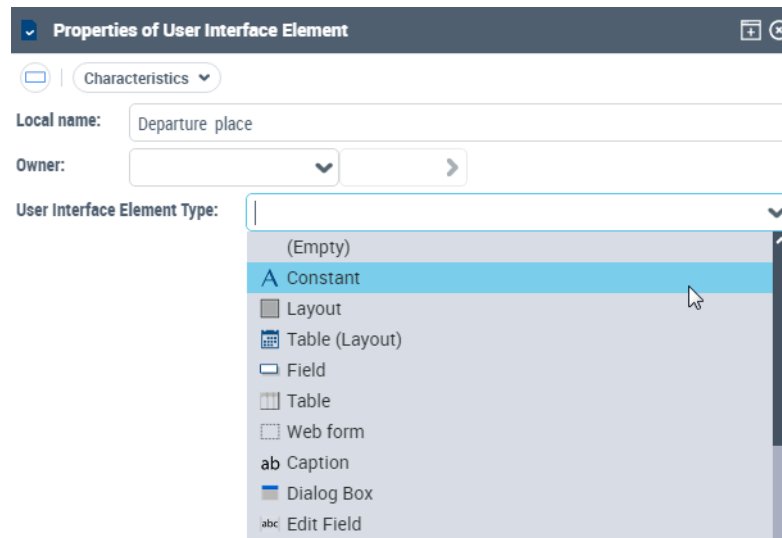
Buttons allow you to modify the appearance of the interface:

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

Connecting the event to an element

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

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



MODELING IT INFRASTRUCTURES



Functionalities proposed by **HOPEX IT Architecture V2** for modeling complex infrastructures enable representation of equipment, IT and organizational resources required for system deployment and operation: interactions between components, communication means supporting these interactions, and services offered and used by the modeled architecture.

The following points are covered here:

- 6 [Describing a Resource Architecture;](#)
- 6 [Describing IT Components.](#)
- 6 [Describing communications in a Technical Infrastructure.](#)

DESCRIBING A RESOURCE ARCHITECTURE

A *resource architecture* comprises equipment, IT and organizational resources required for operation of a complex infrastructure (system).

Communications between these components are represented by interactions and the equipment means supporting these 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.

Creating a Resource Architecture

To create a resource architecture:

1. From the **Infrastructure** navigation pane, select **Infrastructure > Resource Architecture**.
The list of resource architectures appears.
2. Click **New**.
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 list.

Managing a Resource Architecture Assembly Diagram

Creating a Resource Architecture Assembly Diagram:

To create a Resource Architecture Assembly Diagram:

- > Right-click the resource architecture and select **New > Resource Architecture Assembly Diagram**.
The diagram opens in the edit area.

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. You can select an existing resource architecture or create a new one.
 - *To create a new resource architecture, simply enter its name.*
3. Click **OK**.

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.*
 - *In the same way, you can add an org-unit in the Resource Architecture Assembly Diagram.*

Describing Resource Architecture Technical Resources

The network resources group both IT resources, such as servers, networks and devices, and non IT resources such as vehicles for example.

- *For more details on hardware resources, see [Describing IT Components](#).*

Adding an IT infrastructure

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 made up of different IT hardware components such as: IT technical devices, computers or IT networks.*

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 type, simply enter its name.*
 - *In the same way, you can add another hardware resource in the Resource Architecture Assembly Diagram. For more details on hardware resources, see [Describing IT Components](#).*

Channels and communication ports

In a resource architecture, communication 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 [Communication channels](#).

Communication ports enable connection of resource architecture physical assets with external equipment elements.

Describing the Services in a Resource Architecture Assembly Diagram

Describing Resource Architecture Services and Requests

A resource architecture is created to assure one or several services.

- *For more details, see [Service points](#) and [Request points](#).*

Describing Interaction in a Resource Architecture Assembly Diagram

In a resource architecture assembly diagram, **Interactions** enable representation of exchanges between organizational entities.

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

Exchange terms are defined by an **Exchange contract** assigned to the interaction.

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

You can define 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 details on interaction management terms, see [Describing Communications at Equipment Level](#).

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:

1. From the **Infrastructure** navigation pane, select **Infrastructure > Resource Architecture Environments**.
The list of resource architecture environments appears.
2. Click **New**.
The **Creation of a Resource Architecture Environment** window opens.
3. Enter the **Name** of your environment as well as its **Owner** and click **OK**.
The new architecture environment appears in the list.

To create a resource architecture environment diagram

To create a resource architecture environment diagram:

- > Right-click the resource architecture environment and select **New > Resource Architecture Environment Diagram**.
The diagram opens in the edit area.

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, such as a call center, to a resource architecture 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. An addition dialog box prompts you to choose the resource architecture implemented (for example, "eCommerce purchase"). 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, such as a processing center for purchase requests, is implemented in the call center environment, you will add a *partner resource architecture* component type 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 objects toolbar of the resource architecture environment diagram, 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 IT COMPONENTS

Describing an IT infrastructure

) *An IT infrastructure is made up of different IT hardware components such as: IT technical devices, computers or IT networks.*

You can describe the components of an *Infrastructure* in an infrastructure assembly diagram.

Creating an IT infrastructure

To create an IT infrastructure:

1. From the **Infrastructure** navigation pane, select **Infrastructure > IT Infrastructure**.
The list of IT infrastructures appears.
2. Click **New**.
The **Creation of IT Infrastructure** window appears.
3. Enter the **Name** of your infrastructure as well as its **Owner** and click **OK**.
The new IT infrastructure appears in the list.

Creating an Infrastructure Assembly Structure Diagram

To create an infrastructure assembly structure diagram:

- > Right-click the IT Infrastructure and select **New > Infrastructure Assembly Structure Diagram**.
The diagram opens in the edit area.

Using an Infrastructure Assembly Structure Diagram

In an infrastructure assembly structure diagram, you can insert:

You can insert in this diagram:

- *IT servers* and *IT 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.*
 -) *An IT device is a computer that provides a service directly to the end user. This computer can house databases and run applications. This is, for example, a workstation, a laptop or a smartphone.*
- *IoT Devices*,
 -) *An IoT device is both a hardware device and a computing device which provides combined 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 Peripheral Device**,
 -) An IT Peripheral Device can host and run Software Technology. Conjointly with its hosted software, it provides technical services. This consists of, for example, Examples: 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.
- communication ports and channels,
- service and request points,
 - For more details, see [Service points](#) and [Request points](#).
- interactions.
 -) An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.
 - For more details on interactions, see [Describing Communications at Equipment Level](#).

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 more details, see [Infrastructure Description Report](#).

Describing a Computing Device

Accessing the list of computing devices

To access all the different types of computing devices:

1. From the **Infrastructure** navigation pane, select **Infrastructure > Computing Devices**.
2. Select **All Computing Devices** tab.

The list of all computing devices appears:

- **Servers**,
- **IT Devices**,

) An IT device is a computer that provides a service directly to the end user. This computer can house databases and run applications. This is, for example, a workstation, a laptop or a smartphone.

- **IoT Devices**.

Creating a computing device

To create a **computing device**:

1. From the **Infrastructure** navigation pane, select **Infrastructure > Computing Devices**.

2. Select **All Computing Devices** tab.
The list of all computing devices appears.
3. Click **New**.
4. Select the computing device type that you want to create and click **Next**.
5. 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:

- > Right-click the computing device and select **New > Computing Device Assembly Diagram**.
The diagram opens in the edit area.

You can insert the following in a computer assembly diagram:

- *application technical area host*, see [Creating an application technical area host in a computing device assembly diagram](#),
) *An application technical area represents an organizational element of an application according to technical criteria. For example, this can be the user interface or a process. Each application technical area is associated with one or more technologies. The deployment of several application technical areas is necessary for the application to be operation.*
- *software technology host*,
) *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.*
- *micro-service host*,
) *A micro-service is a software component that can be deployed autonomously, but which does not directly provide an end user service. It can interact with other application services, applications or application systems. This is a deployable software component that uses software technologies. For example: an authentication service, a PDF file printing service.*
- *data store host*,
) *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.*
- communication ports and channels,
- service and request points,
- interactions.

Creating an application technical area host in a computing device assembly diagram

To describe that an computing device hosts applications such electronic mail, you will create an **Application Technical Area Host** and associate the required application with it.

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

To create an application technical area host:

1. In the objects toolbar of the computer assembly diagram, click **Application Technical Area Host**.
2. Click in the diagram frame.
An addition dialog box prompts you to select a housed **Application**.
3. Select for example the electronic mail application and click **Add**.
4. Then create the associated application technical area and click **OK**.

Communication channels and ports in a computer assembly diagram

Communication channels support transfer of information between equipment resources. For more details on creation of these channels and the associated communication protocols, see [Communication channels](#).

Communication ports enable connection of IT equipment resources with external equipment elements.

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 **Infrastructure** navigation pane, 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:

- Right-click the IT network and select **New > Infrastructure Assembly Structure Diagram**.

The diagram opens in the edit area.

- *For more details on this type of diagram, see [Creating an Infrastructure Assembly Structure Diagram](#).*

DESCRIBING COMMUNICATIONS IN A TECHNICAL INFRASTRUCTURE

In an infrastructure, communications are based on:

- service points, request points and interaction points for organizational communications,
- communication ports and channels for technical communications.

Describing services and requests

Service points

Services assured by resource architecture 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 is requested according to precise terms defined by an *exchange contract* assigned to the service point.

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

Resources activated to assure a service are linked to the service point by interactions. If activation of several resources is necessary, then several interactions must be created between the service point and the architecture resources.

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

Request points

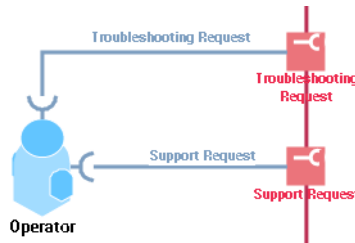
A *request point* is used to represent the use of an external service.

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

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

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

Resources that issue a request are linked to the request point by an 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 more details, see [Creating a Service Point or a Request Point](#).

Describing Communications at Equipment Level

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](#).

Communication channels

Communication Channels enable connection of equipment resources between themselves, with organizational resources or with communication points.

) A communication enables physical connection between two equipment resources. It supports interactions that define communication between these resources. Communication channels connect resources with the exterior via communication ports.

Creating a communication channel

To create a communication channel:

1. In the resource architecture assembly diagram objects toolbar, click **Communication 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 a communication 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.*



USING SERVICE CATALOGS



A service catalog describes the list of functionalities covered by a service as well as the technical or functional elements that implement these functionalities.

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

The following points are covered here:

- 6 [Introducing service catalogs;](#)
- 6 [Populating a service catalog.](#)
- 6 [Service catalog reports.](#)

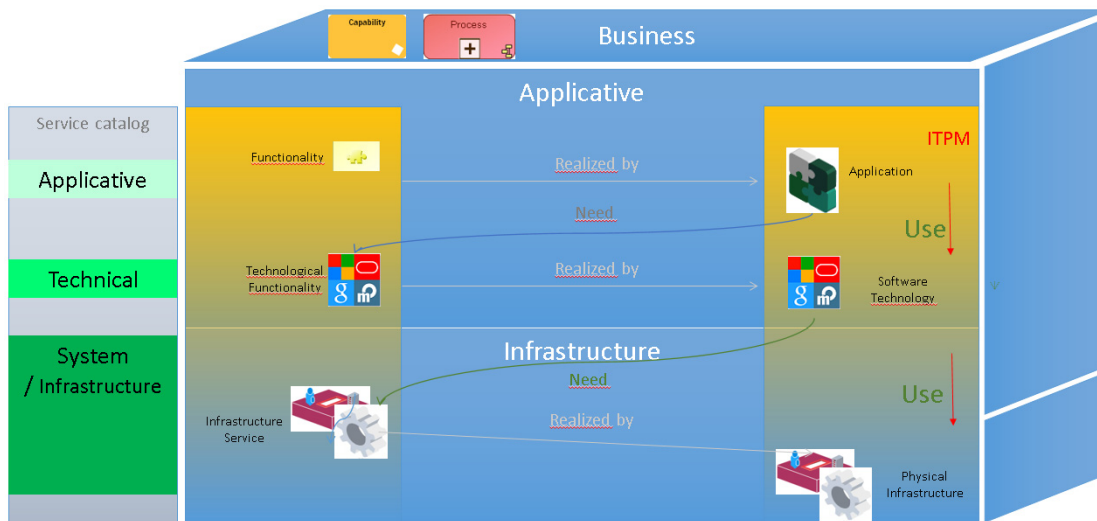
INTRODUCING SERVICE CATALOGS

The types of service catalogs

HOPEX IT Architecture V2 offers the following service catalogs:

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

The figure below presents the positioning of service catalogs offered by **HOPEX IT Architecture V2**.



Implementation of a service catalog

In **HOPEX IT Architecture V2**, a service catalog is made up of service catalog item. For example, an *information service catalog* is made up of several *information service catalog items*.

-) 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.
-) An information service catalog item defines which functionality is in the catalog and which application artifacts provide the functionality.

A *service catalog item* is connected to one or more *functionalities*.

-) A functionality is a service required by an org-unit in order to perform its 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.

With **HOPEX IT Architecture V2**, the implementation of a functionality 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 catalogs according to their category.

Type of service catalog	Type of functionality	Types of object that deliver the service
Business	Features	Business capability
Information	Features	All types of technical and functional objects that implement a functionality with HOPEX IT Architecture V2 . For more details, see Describing a Functionality Map with HOPEX IT Architecture V2 .
Technical	Technical functionalities	All types of technical objects implement a functionality with HOPEX IT Architecture V2 : technical infrastructures, equipment and IT infrastructures, micro services, technical zones, all IT equipment (servers, networks, devices).
hardware	Hardware functionalities	Hardware and IoT Device.

POPULATING 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:

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

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

Creating an information service catalog

To create an **information service catalog**:

1. From the **Catalogs** navigation pane, select **Information service catalogs> Information service catalogs**.
The lists of information service catalogs appear in the edit area.
2. Click **New**.
A new information service catalog appears in the edit area.
3. Enter the **Name** of your catalog and its **Owner**.
 - In the same way, you can create a **technical service catalog** or a **hardware service catalog**.

Accessing the list of service catalogs

To access the list of *information service catalogs*

- > From the **Catalogs** navigation pane, select **Information service catalogs > Information service catalogs**.

The lists of information service catalogs appear in the edit area.

A service catalog is defined by all of its *information service catalog items*.

) *An information service catalog item defines which functionality is in the catalog and which application artifacts provide the functionality.*

To access the list of *information service catalog items*:

1. From the **Catalogs** navigation pane, select **Information service catalogs > Information service catalog items**.

2. In the **Information service catalog** field, select the catalog that interests you.

The tree of information service catalog item appears in the edit area.

Adding a service catalog item

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

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

To add an *information service catalog item*:

1. Open the **Characteristics** properties page of an information service catalog.

2. In the **Information Service Catalog items** section, click **New**.

In the window that opens, you can select an existing functionality or create a new one.

3. Select the functionality that interests you and click **OK**.

The catalog element appears in the list with the name of the associated functionality.

Specifying the implementation of a service catalog item

To describe all the Service Catalog elements that implement a functionality, you will define a *Realization*.

) *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 describe that a functionality is implemented by a Service Catalog element:

1. Open the **Characteristics** properties page of an information service catalog.

2. In the **Information Service Catalog items** section, click **New**.

An add functionality dialog box appears:

3. Select the functionality that interests you.

4. Click **OK**.

The functionality realization appears in the properties page.

SERVICE CATALOG REPORTS

With a report, you can determine, for a list of service catalogs, the list of functionalities covered, and for each of them, the implementation means.

Report parameters

This consists of defining report input data.

Parameters	Parameter type	Constraints
List of service catalogs	All types of service catalogs: - information, - technical, - hardware.	A mandatory catalog.

Creating a service catalog report

To create a service catalog report:

1. From the **Catalogs** navigation pane, select **Reports**.
A new report opens in the edit area.
2. In the **Parameters** section, in the **Service Catalog** field, select the service catalogs that you want to present in your report.
3. Click **Refresh the Report**.
The new report appears.

USING IT ARCHITECTURE DIAGRAMS



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

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

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

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

STRUCTURE DIAGRAMS AND FLOW SCENARIO DIAGRAMS

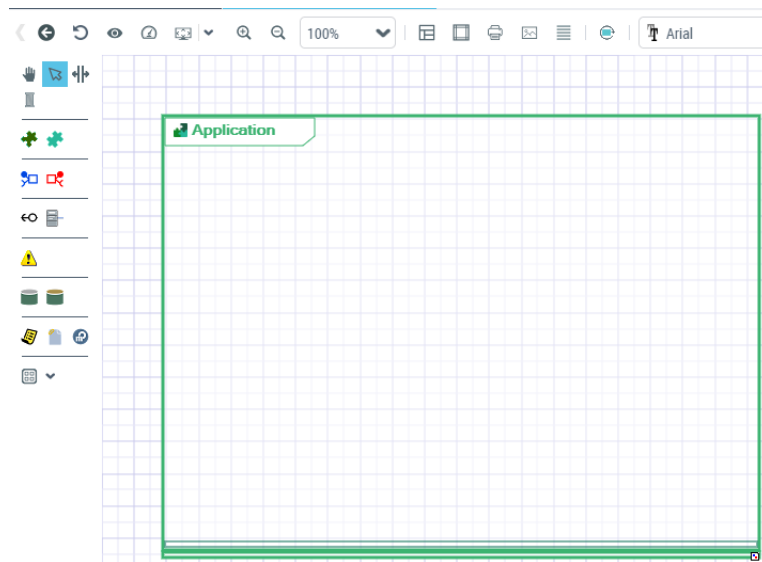
INITIALIZATION

To create an Application Structure Diagram, for example:

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

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

Example



HOPEX IT Architecture V2 diagrams initialization

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

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

Icon	Description
	Refresh diagram
	Properties of the diagram Provides access to the diagram properties
	Reinitializing components Add the described object components in the diagram.

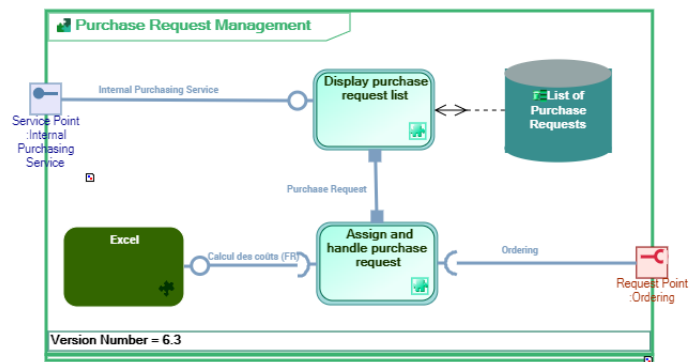
CREATING A STRUCTURE DIAGRAM

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

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

Creating an Application Structure Diagram

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



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

The components of an Application Structure Diagram

An Application Structure Diagram includes:

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

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

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

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

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

In the example, this is the Excel application.

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

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

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

Adding an application service to an application structure diagram

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

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

DESCRIBING A SCENARIO OF FLOWS

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

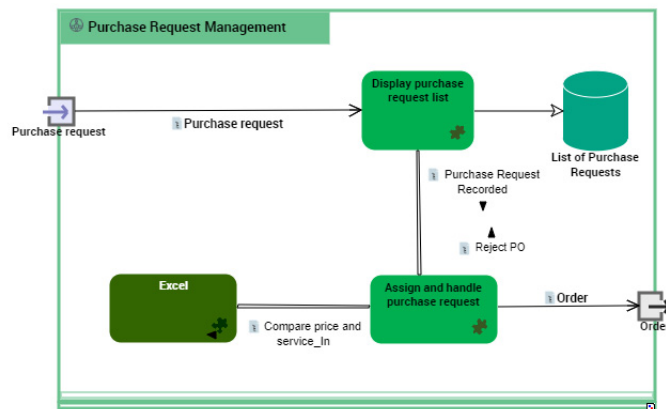
Depending on the solutions available to you, two types of diagram are proposed:

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

Using a Scenario of Application Flows Diagram

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

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



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

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

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

The interactions offered between these elements:

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

Creating a Scenario of Application Flows diagram

To create a scenario of application flows:

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

Adding an IT service to the scenario of application flows

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

To add an **IT service**:

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

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

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

Managing application flows in a scenario of application flows

Creating an application flow with content

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

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

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

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

To create the *application flow*:

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

Creating an application flow channel

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

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

To create an *application flow channel*:

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

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

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

Reinitialize components in a scenario of flows

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

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

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

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

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

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

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

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

1. In the scenario objects toolbar, click **Local Application Data Store**.
2. Click in the described application frame.
A creation window prompts you to choose the *application data area* that represents the physical structure that will concretely support the application data store.

) *An application data area represents a set of data stored in entity/relationship format and used in the technical architecture of an application.*

- For more information on data stores, see "Logical and application data areas" chapter in **HOPEX Information Architecture** guide.

3. Select the existing **Application data area** that interests you.
4. Click **OK**.
The local application data store appears in the diagram with the name of the physical data domain selected.

Creating an application data channel

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

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

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

An application data channel appears in the diagram.

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

Using a Flow Scenario Sequence Diagram

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

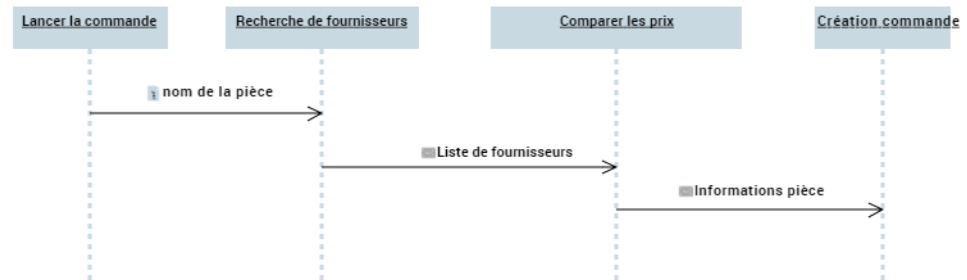
This type of diagram can be built for an application environment, an application, an application system, an IT service or a micro-service.

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

A flow scenario sequence of flow diagram contains:

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

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



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

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

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

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

Using a Flow Scenario Sequence Diagram

To create an application environment scenario sequence:

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

Instances of applications, IT services or interfaces

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

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

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

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

To create an application service instance for example:

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

Message instance

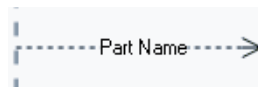
Message instances define the data exchanged between application instances, application services and the interfaces. The sequence described in the flow scenario sequence diagram indicates the message sending order.

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

To create a message instance:

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

The message instance exchanged between the two objects is drawn.

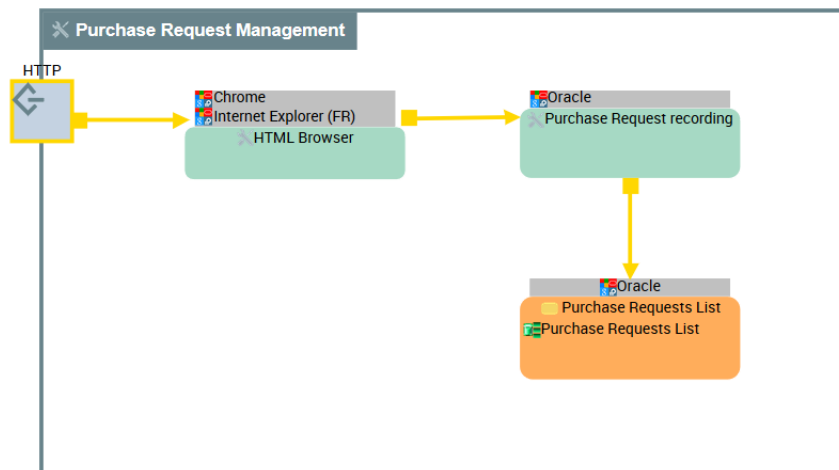


DESCRIBING A TECHNICAL ARCHITECTURE

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

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

The technical communication line is based on http.



Technical Architecture Example

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

Creating an Application Technical Architecture Diagram

Elements represented in an Application Technical Architecture diagram are:

- *application technical areas*,

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

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

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

Adding an application technical area to an application technical architecture diagram

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

To create an **application technical area**:

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

You can add data technical areas in the same way.

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

Defining the software technologies used by an application technical area

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

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

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

Describing technical communications

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

) A technical communication line represents a technical connection between architectures or application technical areas through client and server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.

) A communication protocol is a set of standardized rules for transmission of information (voice, data, images) on a communication channel. The different layers of protocols can handle the detection and processing of errors, authentication of correspondents, management of routing.

Creating a technical communication line

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

To create a **technical communication line**:

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

Technical ports

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

) A technical server port is a point used to open communications with a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).

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

Technical ports comply with network application protocols.

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

DESCRIBING DATA EXCHANGES



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

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

MANAGING INTERACTIONS

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

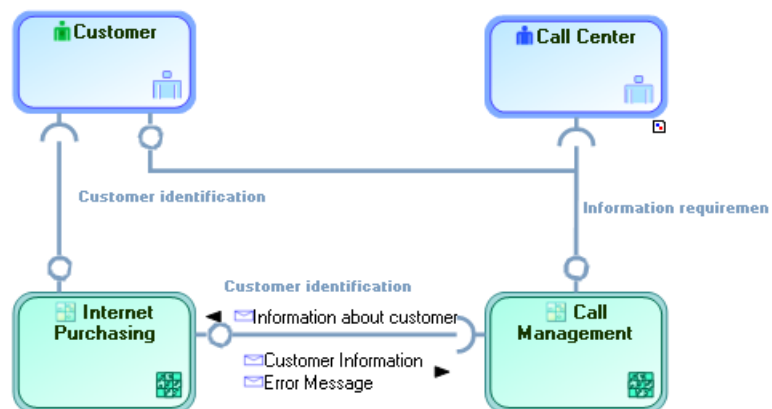
) An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described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

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

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

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

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



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

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

Creating an Interaction

To create an interaction:

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

Describing Service and Request Points

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

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

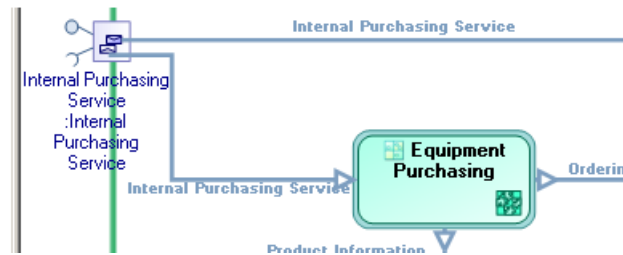
Service points

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

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

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

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



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

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

Request points

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

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

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

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

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



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

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

Creating a Service Point or a Request Point

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

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

To create a service point:

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

To change the service point name:

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

Defining the Element Interaction Point

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

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

Characterizing the element interaction point

To modify the properties of the element interaction point:

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

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

DESCRIBING EXCHANGES

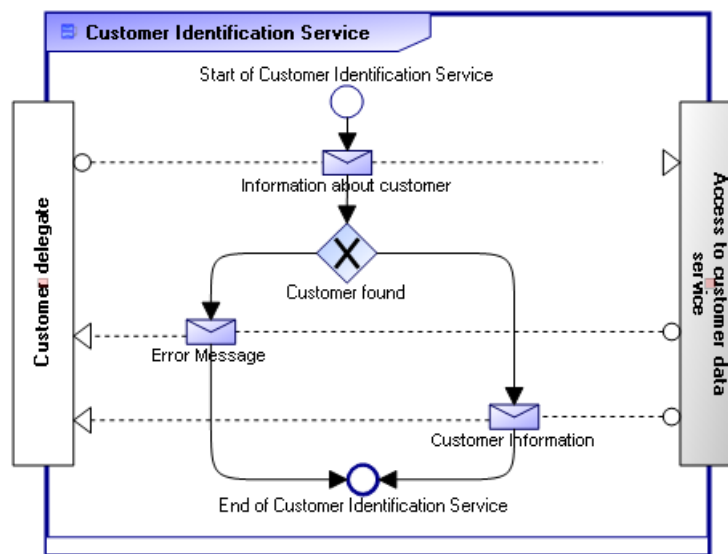
Content of an interaction is described by an *exchange contract*.

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

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

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

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



"Customer Identification Service" Exchange Diagram

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

Creating an Exchange

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

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

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

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

The Creation of Exchange Use dialog box opens.

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

Describing Exchanges

Creating an exchange diagram (BPMN)

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

To create an exchange diagram:

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

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

Creating a message flow with content

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

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

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

To create a message flow and its content:

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

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

Managing events, gateways and sequence flows

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

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

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

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

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

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

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

DESCRIBING EXCHANGE CONTRACTS

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

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

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

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

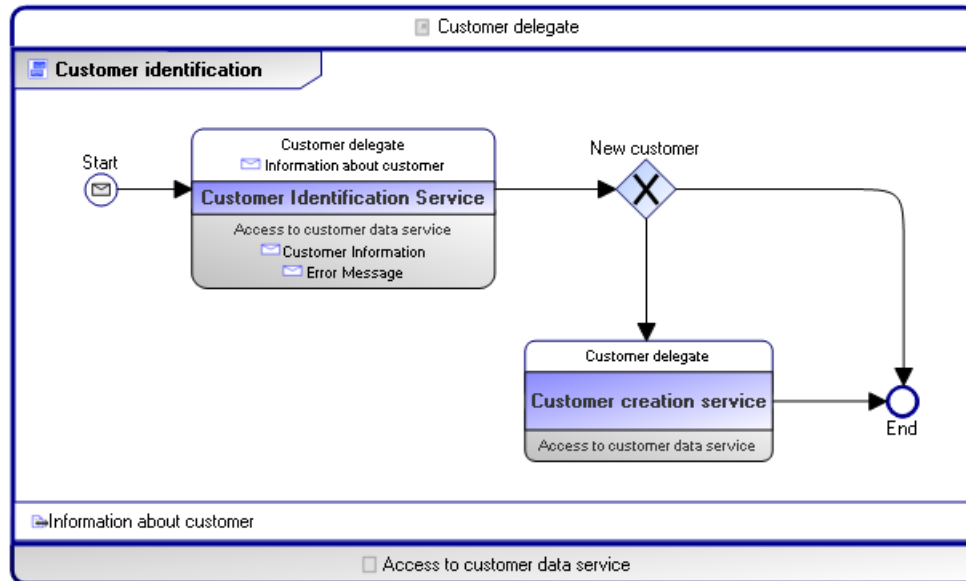
An exchange contract is described by a sequence flow of operations which are represented:

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

Examples of Exchange Contract Diagrams (BPMN)

Exchange contract diagram (BPMN)

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



Exchange Contract Diagram (BPMN) "Customer Identification"

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

Progress steps are represented by *exchange use*.

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

Advanced communication exchange contract example

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

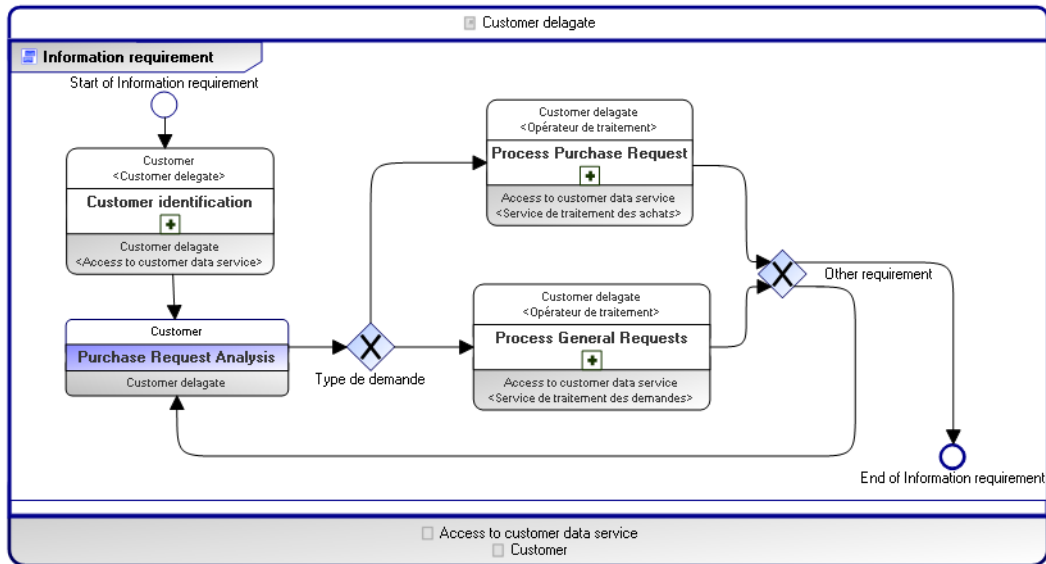
- either by *exchange use*
- or by *exchange contract use*

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

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

- customer/supplier, or
- sender/recipient

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



"Information Requirement" Exchange Contract Diagram (BPMN)

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

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

Creating an Exchange Contract from an Interaction

You can also create a new exchange contract:

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

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

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

Creating an Exchange Contract Diagram (BPMN)

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

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

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

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

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

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

Defining an Exchange or an Exchange Contract Use

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

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

To create an *exchange contract use*:

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

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

HOPEX IT ARCHITECTURE V2 REPORTS



HOPEX IT Architecture V2 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 V2** offer various analysis presentation possibilities. Some reports are shared with other solutions, for example **HOPEX Business Architecture & Strategic Planning**.

Several types of reports are available on several types of object.

- 6 Exploded diagram report
 - For more details, see "Launching the exploded diagram report" in the diagram chapter of **HOPEX Common Features** guide.
- 6 **Building Block Breakdown report**
 - For more details, see [Building Block Breakdown report](#).

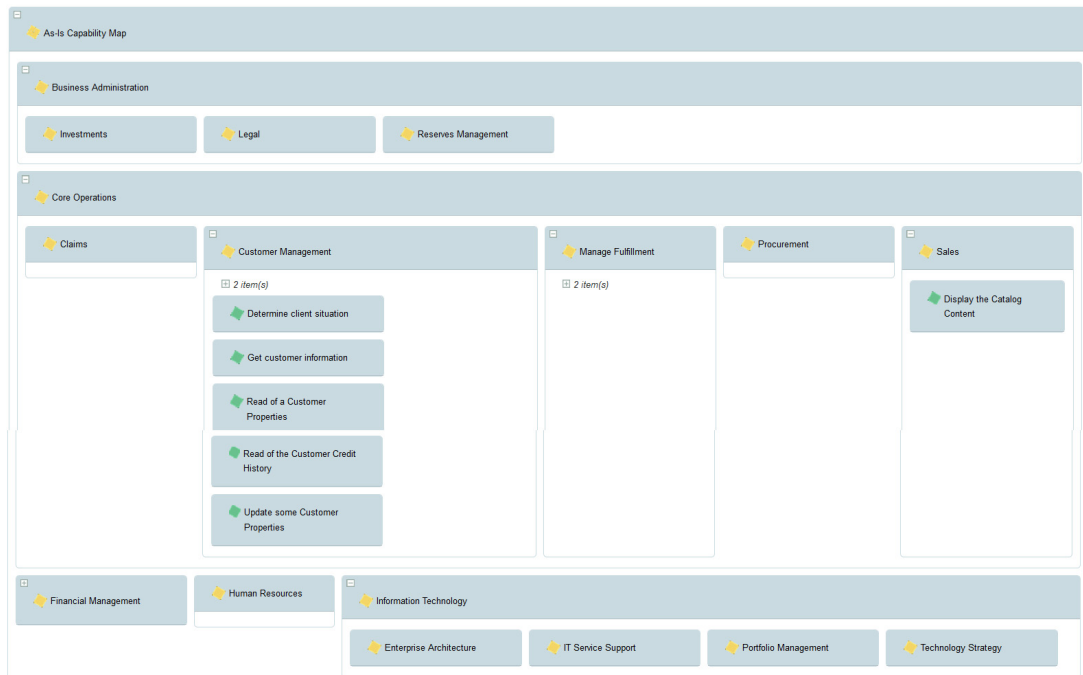
BUSINESS CAPABILITIES REPORTS

Breakdown map of business capabilities

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.

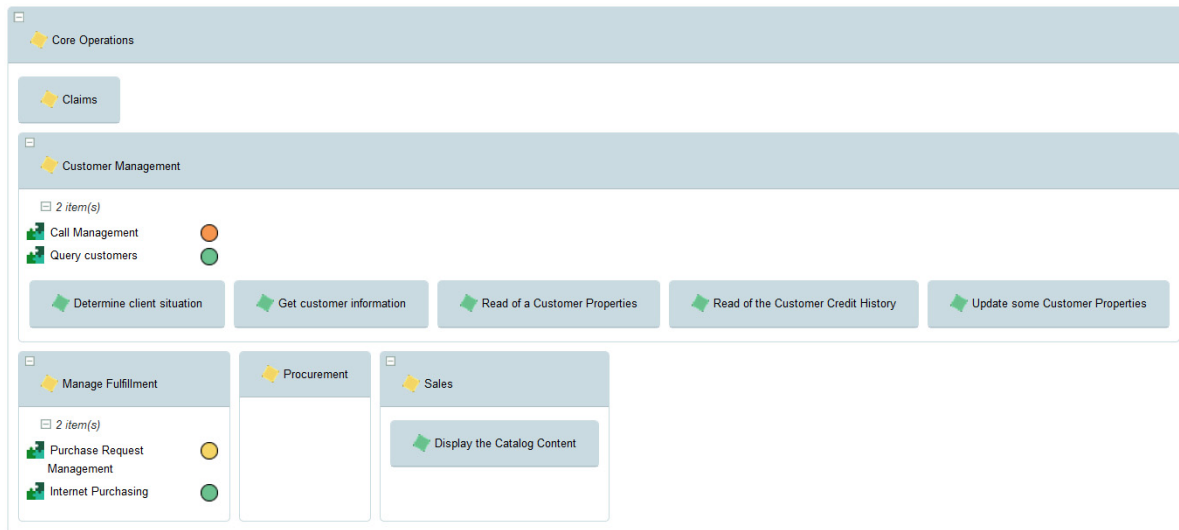
Report examples

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	Capability map, Capability	One object mandatory.

Business Capabilities and Expected Functionalities Matrix

With the **Business Capabilities and Expected Functionalities Matrix** Report DataSet Definition, you can create a matrix-type instant report to view links between functionalities and business capabilities.

Report examples

The example below enables viewing, in the form a table, of the coverage rate of the capability specified as parameters.

Matrix		Bar Chart	
		Manage invoices	Print the Invoices
Accounting			
Billing		✓	✓
Enterprise Architecture			
IT Service Support			
Portfolio Management			
Technology Strategy			

- For more details on how to associate a business capability with a functionality, see [Defining the functionalities associated with business capabilities](#).

REPORTS ON THE ARCHITECTURE FUNCTIONAL COVERAGE

Building Block Breakdown report

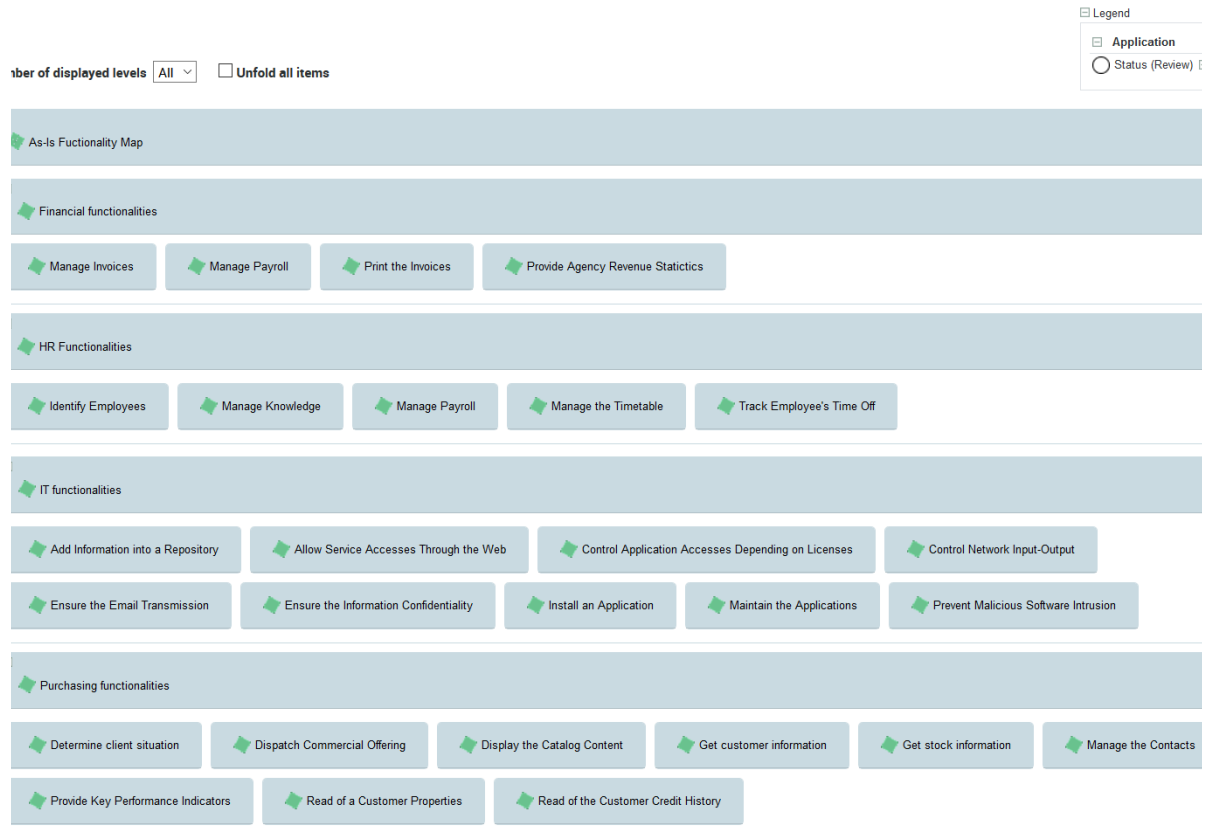
You can use this report to display the realization coverage of business capability elements by operational elements such as logical and physical applications, application systems, etc.

- 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 the, chapter "Handling a Breakdown Report" in the **HOPEX Common Featuresguide**.

Report examples

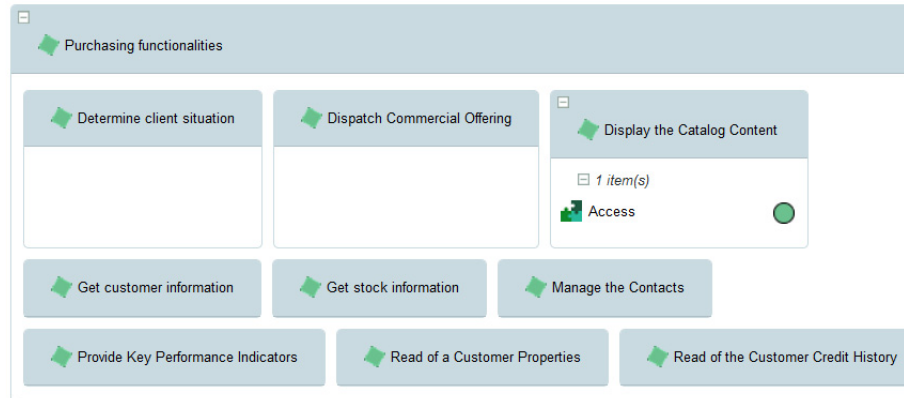
The example below enables viewing of the functional breakdown of the functionality map specified as parameters.

Building Block Breakdown Report



- Example of functionality breakdown report.

In the example below, the applications that implement the functionalities are presented.



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)

Parameters	Parameter type	Constraints
Color palette	HOPEX palette	Mandatory. The palette delivered by default is "BoxInBox Report Monochrome Grey"
EA Level	Multiple choice: - business function level, - organizational level, - application level, - technical level.	Define which objects of which type of architecture level are displayed for capability realizations; <i>For example, activation of the "applications level" displays the business capability realizations for the Application System Environment, the Application Systems or the Applications</i>
EA dimension	Multiple choice: - capability models, - agent models, - process model, - information models, - performance models, - results models	Define which types of objects are examined within the framework of the breakdown analysis <i>For example, activation of "capability models" will display the business skills or functionalities required by the capabilities that are broken down</i>

Matrix of Business Capabilities and Expected Functionalities

You can use this report DataSet to view links between functionalities and business capabilities.

Report examples

The example below enables viewing, in the form a table, of the coverage rate of the capability specified as parameters.

Matrix		Bar Chart
		Manage invoices Print the Invoices
Accounting		
Billing	✓	✓
Enterprise Architecture		
IT Service Support		
Portfolio Management		
Technology Strategy		

- For more details on how to associate a business capability with a functionality, see [Defining the functionalities associated with business capabilities](#).

APPLICATION ARCHITECTURE REPORTS

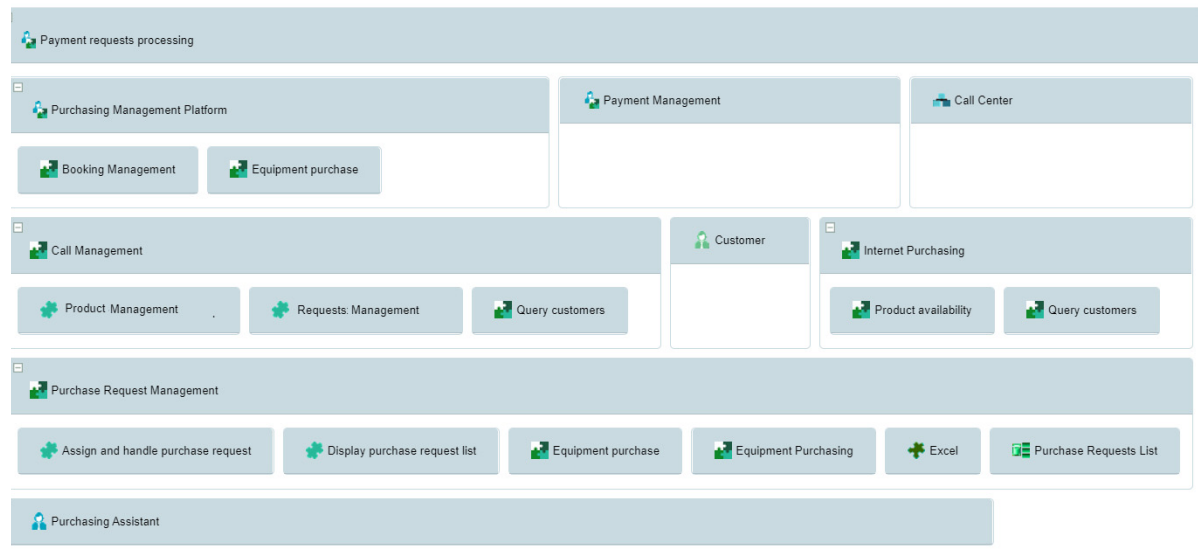
Application Architecture Breakdown Report

Based on the same principle as the breakdown reports, this report presents the breakdown of a logical application architecture with respect to its components.

- For more details on use of a breakdown report, see the, chapter "Handling a Breakdown Report" in the **HOPEX Common Featuresguide**.

Report example

The example below shows the breakdown of the application system area for purchasing requests.



- Example of a application system breakdown report

Report parameters

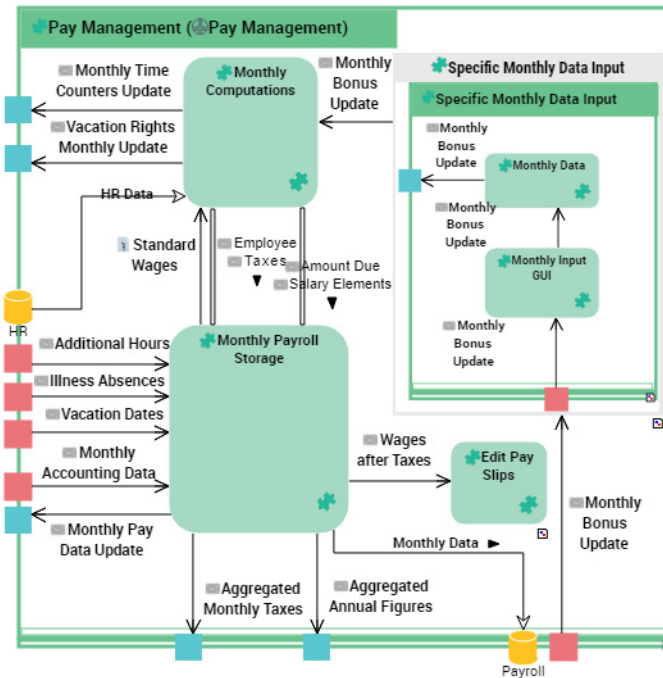
This consists of defining report input data.

Parameters	Parameter type	Constraints
Root object	application architecture building block	One object mandatory.

Impact report (Scenario)

This report is used to analyze the impact of the failure of an application component on the application systems that use it or with which it exchanges application flows in the **use scenario** contexts of this application component.

For example, the "Specific Monthly Data Input" IT Service used in the context of the scenario of flow diagram below.



The report presents the list of application flows exchanged with the flow scenario components presented below.

1. Impact (Scenario)

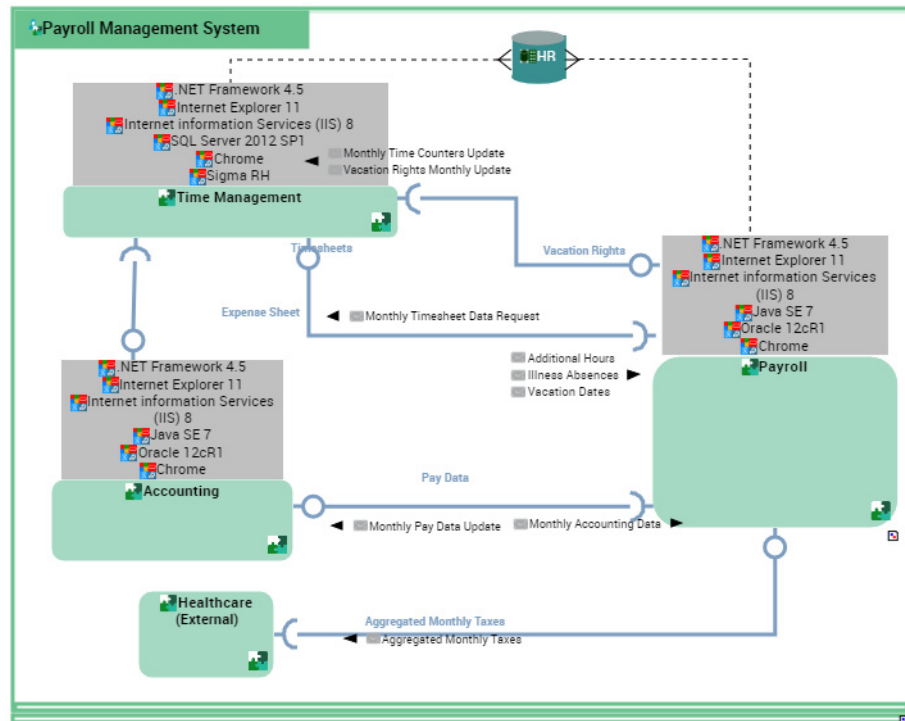
Specific Monthly Data Input

Specific Monthly Data Input	
Payroll	Specific Monthly Data Input Bonus in Pay Slips Management Bonus in Pay Slips Management
Monthly Computations	Monthly Bonus Update Monthly Bonus Update Monthly Computations Pay Management
Payroll	Monthly Computations Bonus in Pay Slips Management Bonus in Pay Slips Management
Manager	
Payroll Management System	Manager Bonus in Pay Slips Management Bonus in Pay Slips Management

Impact report (Structure)

This report is used to analyze the impact of the failure of an application component on the application components that use it or with which it interacts using exchange contracts as described in its **structure diagrams** that use this application system.

For example, the "Time Management" application is used in the context of the structure diagram below.



The report presents the list of components with which the application interacts using the exchange contracts.

1. Impact (Structure)

Time Management

Time Management	
Payroll Management System	Time Management
Payroll Management System Environment	Payroll Management System
Payroll	Timesheets Timesheets Payroll Management System
Payroll Management System	Payroll
Payroll Management System Environment	Payroll Management System
Accounting	Expense Sheet Expense Sheet Payroll Management System
Payroll Management System	Accounting
Payroll Management System Environment	Payroll Management System

INFRASTRUCTURE REPORTS

Infrastructure Description Report

This report makes it possible to analyze the components of the infrastructure and the expected relationships between them.

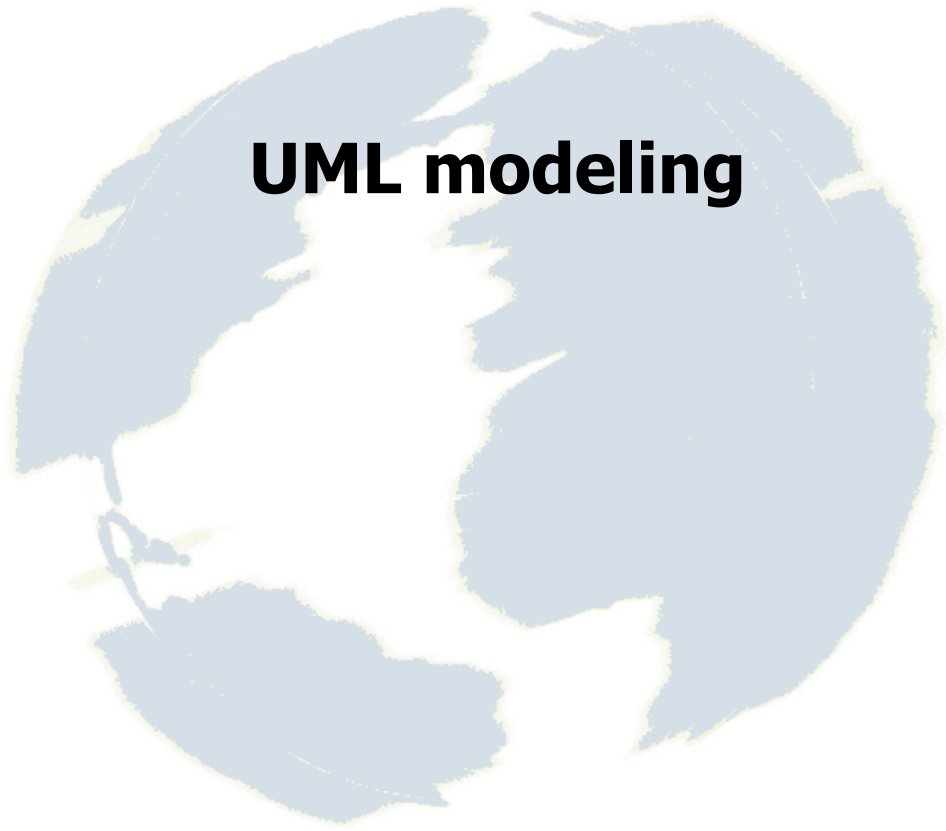
- For more details, see [Describing an IT infrastructure](#).

The different chapters of this report verify that each element described in an infrastructure assembly structure diagram hosts a software or hardware component.

The chapters of this report are devoted to:

- the list of networks,
- the list of computing devices,
- the list of IT technical devices,
- the list of communication channels.

UML modeling



ABOUT UML IMPLEMENTATION



UML (Unified Modeling Language) is established as the standard for the graphic modeling of information systems. **HOPEX IT Architecture V2** offers a set of tools allowing you to model your IS in compliance with version 2.3 of this standard.

- *To access these functions, you must be connected with **UML Designer**.*

The aim of this part is to introduce you to the main UML functionalities provided by **HOPEX IT Architecture V2**.

- 6 ["Overview", page 218](#)
- 6 ["Organization of UML Diagrams", page 220](#)

OVERVIEW

The facilities provided by **HOPEX IT Architecture V2** 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 ["Use Case Diagram", page 223](#).

Identifying objects

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

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

See ["The Class Diagram", page 235](#).

Describing behaviors

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

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

See:

- ["State Machine Diagram", page 297](#)
- ["Activity Diagram", page 307](#)

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 315](#).

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 286](#).

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 289](#)
- ["Composite Structure Diagram", page 293](#)

Specifying deployment

Implementation of objects in an concrete working environment can be specified in the **deployment diagram**.

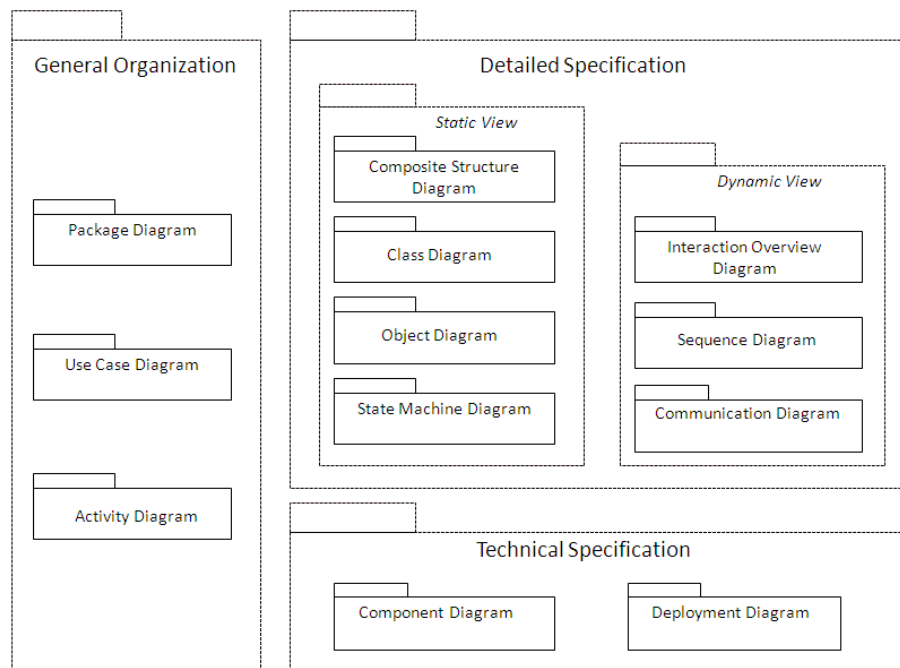
See ["The deployment diagram", page 331](#).

ORGANIZATION OF UML DIAGRAMS

General organization

Use case diagrams show the main interactions between the system being analyzed and its environment, and indicate its main sub-systems.

Package diagrams provide a more technical breakdown of the system. Dividing a system into packages imposes some structure, as an object can only be in one package. You can begin drawing package diagrams as soon as you have identified the main components of your system (Sales, Production, Invoicing, etc.).



Detailed specification

The main diagram is the **class diagram**. It describes the essential semantics of the objects in the system. This is where designers will generally spend most of their time. Classes are generally discovered by iteration between class and sequence diagrams.

The **state machine diagram** describes the static aspects of an object: the different states it can be in and the possible state transitions. This fleshes out the class description.

The **interaction diagrams** describe the dynamic aspects of the system, by showing the interactions between objects. They provide a detailed description of the different scenarios in a use case. The sequence diagram specifies how a scenario progresses

over time, while the communication diagram stresses the interactions between objects.

Technical specification and deployment

The **component diagram** describes the different technical components of an application and shows their interactions.

The **composite structure diagram** specifies collaborations between components or component elements in execution of a common task.

The **deployment diagram** is used to specify the system architecture, indicating the workstations or nodes in the information system where the different application components are to be installed.

UML diagram entry points

Diagram	Entry points
Class diagram	Package, class, use case
Object diagram	Class, component, package, use case
Component diagram	Component, package
Composite structure diagram	Component, class, collaboration
Deployment diagram	Package
Package diagram	Package, library
Use case diagram	Package, Use case, Application environment (ADES)
Sequence diagram	interaction
Communication Diagram	interaction
Interaction	interaction
overview diagram	
Activity diagram (UML2)	Activity
State machine diagram	State machine, , protocol state machine

In **HOPEX IT Architecture V2**, the entry points above are accessible in the **Software Design >OO Implementation (UML)** navigation pane.



USE CASE DIAGRAM



The use case diagram constitutes a first step in description of an information system. It enables identification of the functionalities to be provided by the system to meet the requirements of organization actors; it therefore describes interactions between the system and the actors.

The following points are covered here:

- 6 ["Creating a Use Case Diagram", page 224](#)
- 6 ["Use Case Diagram Elements", page 225](#)

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 V2**, you can also create a use case diagram for the application environment of a project. See ["Creating an Application Environment Use Case Diagram"](#).

Creating a Package

To create a package with **HOPEX IT Architecture V2** from the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit area, click **Packages**.
3. In the edit area, click **New**.
The Creation of Package dialog box appears.
4. Enter its **Name**.
5. 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.*
6. Click **OK**.

The package is created and added to the list of packages.

Creating the Use Case Diagram of a Package

To create a use case diagram:

1. Click the icon of the package concerned and select **New > Use Case Diagram**.
The diagram opens in the edit window. The frame of the package is positioned within the drawing.

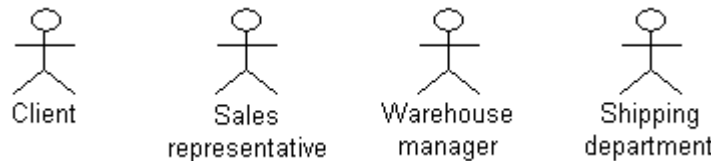
USE CASE DIAGRAM ELEMENTS

- 6 "Actors", page 225
- 6 "Use Cases", page 226
- 6 "Packages", page 226
- 6 "Participations", page 227
- 6 "Use Case Associations: Extensions and Uses", page 229
- 6 "Generalizations", page 232
- 6 "Interfaces", page 233

Actors

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

Examples of actors:



To create an **actor** in a use case diagram:

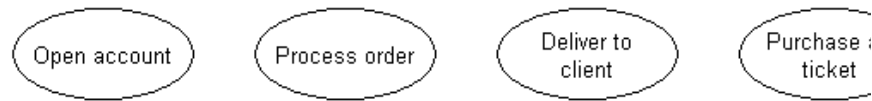
1. In the object toolbar, click **Actor** 
2. Click in the diagram.
The **Add Actor** dialog box opens.
3. Enter the name of the actor, "Receptionist" in this example.
4. Click **Add**.
The actor then appears in the diagram.

M You can create several elements successively without clicking in the toolbar each time by double-clicking the **Actor** button.

M To return to normal mode, press <Esc>, or click on another button in the toolbar.

Use Cases

Examples of *use cases*: processing an order, delivering to a client, opening an account, sending an invoice, establishing credit, purchasing an airline ticket, etc.



To create a use case in a diagram:

1. In the use case diagram objects toolbar, click the **Use Case**  button. The **Add Use Case** dialog box opens.
2. Enter the use case **Name** and click **Create**.

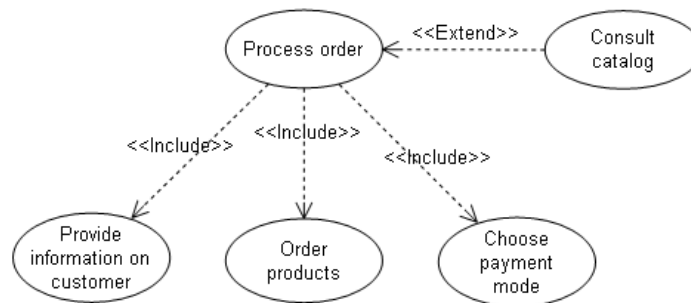
The use case appears in the diagram.

Zooming in on a use case

To open the diagram that describes a use case directly from the package diagram:

1. Right-click the use case.
2. Select **Use Case Diagram**.

The use case diagram opens.

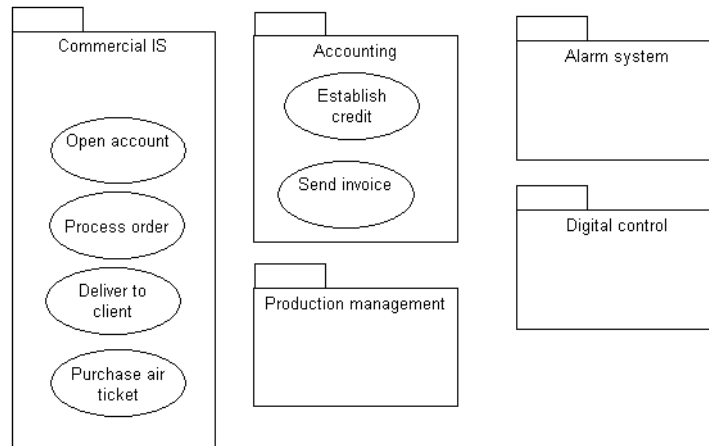


M Zooming in on the description of an object is possible for all elements described by a diagram.

Packages

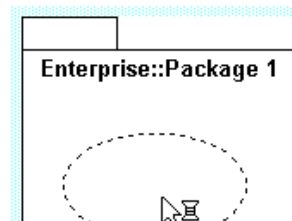
) A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.

Examples of **packages**: the commercial information system, accounting, production management, digital control of a machine, an alarm system, etc.



You can create a package using the **Package**  button in the toolbar. You can then increase its size in order to place use cases within it.

You can link a use case to a package simply by placing it within the package. When you have moved the object within the package, the package shape is highlighted to indicate that the object will be connected to it



- If the linked objects disappear under the package, click the package and select the **Send to Back** button in the Edit toolbar.

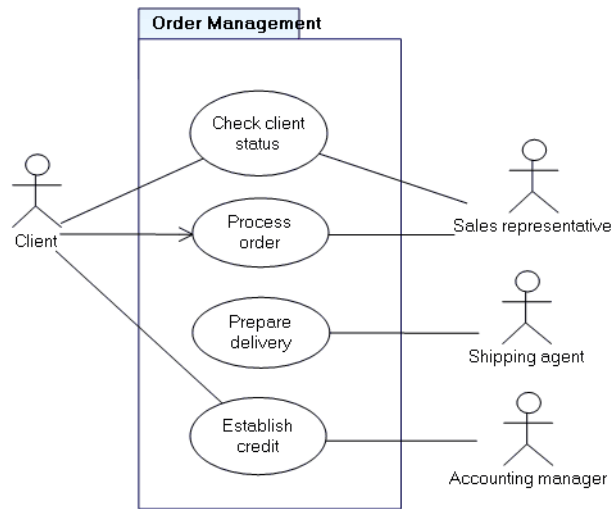
When you move a use case from one package to another using the mouse, it is unlinked from the first one and linked to the second. When you move it with the keyboard arrows, however, the links remain unchanged.

Participations

You can indicate which actor participates in each of the different use cases.

-) A participation indicates that an actor plays a role in a use case.

Examples of participation



- The sales representative participates in order processing and in checking client status;
- The shipping agent participates in delivery;
- The accounting manager participates in setting up loans, etc.

Creating participations

To create a *participation* in a use case diagram:

1. In the insert tool bar, click the **Participation** button 
2. Click the actor concerned, and drag the mouse to the use case before releasing the button.
A dialog box appears:
3. Enter the name of the participation and indicate if the actor is the initiator.
 - It is possible to specify the beginning of the use case by selecting the **IsInitiator** check box in the properties dialog box of the corresponding participation. An arrow appears on the line representing the participation.
4. Click **OK**.

The link representing the participation appears in the diagram.

P A participation is represented by a link, but it is in fact an object with its own properties.

- The spool  is not used to create participations. It is used to create certain types of links, such as those between packages and other objects.
- If you make a mistake, you can delete an object by right-clicking it and selecting the **Delete** command in its pop-up menu. You can also delete a link by right clicking on it and selecting **Delete** or **Disconnect** from the link pop-up menu.

Multiplicities of a participation

Multiplicity can be specified on a participation:

- From the actor, to indicate that several instances of the actor participate in the same instance of the use case (example: participants in a meeting).
- From the use case, to indicate that the same instance of the actor participates in several instances of the use case (example: a sales representative processes several orders from the same customer).

To define multiplicities on a participation:

1. Select the activity concerned and display its **Properties**.
2. In the properties page that opens, click the drop-down list and select **Characteristics**.
A first section allows you to define multiplicity of the actor, a second frame that of the use case.

Having been defined, the multiplicities appear in the diagram.

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.

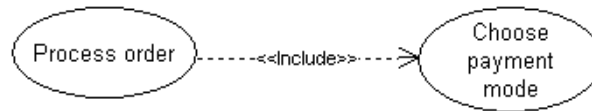
Inclusion relationship

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".



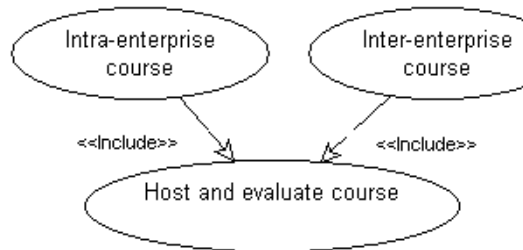
Examples of inclusion

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

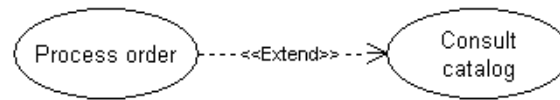
Extend Relation

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:

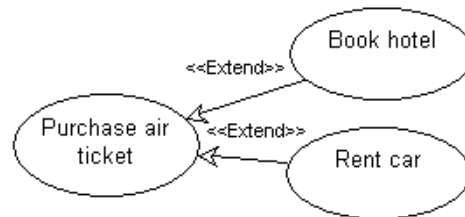
1. In the use case diagram, click the arrow associated with the **Link** button and click **Extension**.
2. Click the use case, for example "Consult Catalog" and drag the mouse to the extension case, for example "Process Order" before releasing the mouse button.
The Creation of Extension dialog box appears. 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.

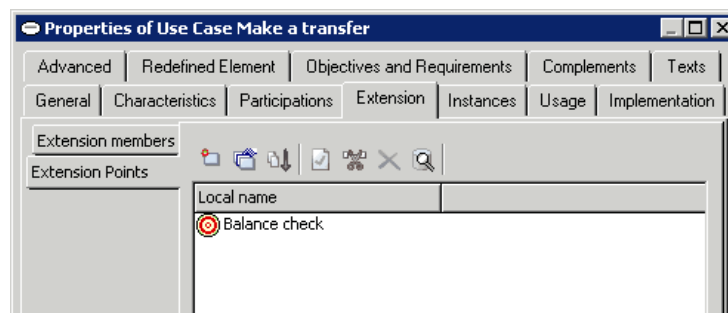


Extension point

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.

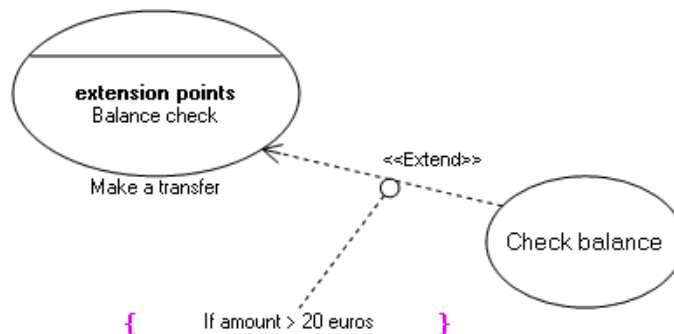


An extension point can be associated with a **constraint** which indicates the moment at which the extension intervenes. You can add a constraint at creation of the extension or later, in the extension properties dialog box.

) A constraint is a declaration that establishes a restriction or business rule generally involving several classes.

Extension point example

The following example presents the use case of a bank transfer; above a sum of 20 euros, customer credit check is triggered.

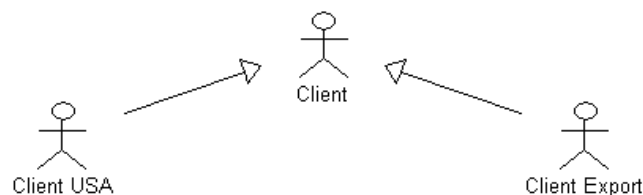


Generalizations

) A generalization represents an inheritance relationship between a general class and a more specific class. The more specific class is fully consistent with the more general class and inherits its characteristics and behavior. However, it also contains additional information. Any instance of the more specific class is also an instance of the general class.

The concept of **generalization** was initially used for classes, but has been extended to other UML concepts such as actor and use case.

Examples of generalizations between actors:

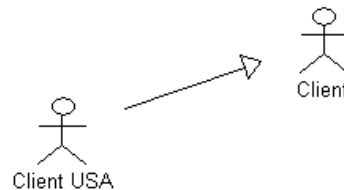


The "Client" actor can have the USA or Export specialization.

To create a generalization between actors in a use case diagram:

- > Click the  button and drag the link from the specialized actor (eg: USA client) to the more general actor (eg: Client).

The generalization then appears in the drawing.



- In the same way you can create a generalization between two use cases.

When creating a second generalization, a dialog box allows you to reuse the existing generalization if it involves the same subject.

Interfaces

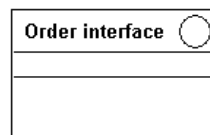
It is possible to complement the description of a use case or actor by describing the *interfaces* by which it communicates with its environment.

Creating an Interface

To create an interface in a use case diagram:

1. In the diagram objects toolbar, click the **Interface** button. 
 - If the **Interface** button does not appear in the toolbar, select **View Views and Details** and select **Classes**.
2. Click in the diagram.
3. In the dialog box that appears, enter the name of the interface and click the **Add** button.

The interface appears in the diagram.



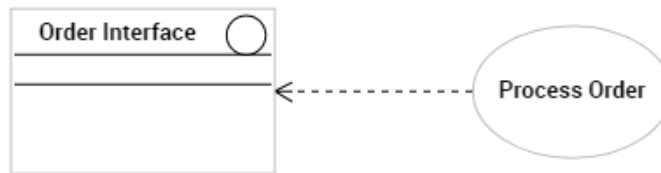
Connecting an interface to a use case

When you connect an interface to a use case, you must specify if it is a supported interface or an interface required by the use case.

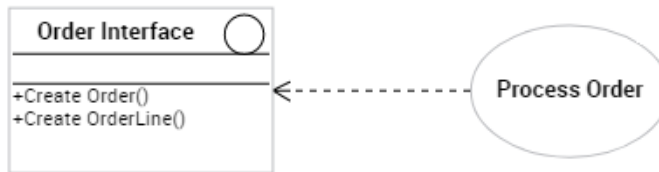
To specify the type of link between an interface and a use case:

1. Click **Connect**  and drag the link from the use case (eg: Process Order) to the interface (eg: Order Interface).
A dialog box appears:
2. Indicate the type of link to be created.
 - Required interface
 - Supported interface
3. Click **OK**.

The link then appears in the diagram.



You can detail which operations the use case can carry out via this interface.



- 6 For more details on required and supported interfaces, see ["Linking interfaces to other objects", page 290](#).

THE CLASS DIAGRAM



A class diagram is used to represent the static structure of a system, particularly the types of objects manipulated in the system, their internal structure, and the relationships between them. An object diagram provides examples to illustrate a class diagram.

The class diagram specification is often considered the most important part in the modeling of an information system. The following points are covered here:

- 6 [Presentation of the Class Diagram](#)
- 6 [Creating a Class Diagram](#)
- 6 [Classes](#)
- 6 [Attributes](#)
- 6 [Operations](#)
- 6 [Signals](#)
- 6 [Associations](#)
- 6 [Generalizations](#)
- 6 [Specifying Interfaces](#)
- 6 [Specifying Dependencies](#)
- 6 [Specifying Parameterized Classes](#)
- 6 [Constraints](#)
- 6 [Object Diagram](#)

PRESENTATION OF THE CLASS DIAGRAM

A class diagram is used to represent the static structure of a system, particularly the types of *objects* manipulated in the system, their internal structure, and the relationships between them.

) *An object is an entity with a well-defined boundary and identity that encapsulates state and behavior. Its state is represented by the values of its attributes and its relationships with other objects. Its behavior is represented by its operations and methods. An object is an instance of a class.*

Examples of objects:

- Business objects:
 - John Williams, Elizabeth Davis and Paul Smith are instances of the "person" class.
 - Orders 10533 and 7322 are instances of the "order" class.
 - SPD-1730 Monitor is an instance of the "item" class.
- Technical objects used for programming:
 - Dlg_Order_Create, Dlg_Client_Query are instances of the window class.
 - Str_Client_Name, Str_Product_Comment are instances of the "string" class.

Data modeling consists of identifying the classes representing the activity of the company, and defining the associations existing between them.

The classes and associations comprising the class diagram associated with a business area of the company must provide a complete semantic description.

In other words, one should be able to describe the activity of a company by using only these classes and associations.

This does not mean that each word or verb used in the explanation maps corresponds directly to an object in the class diagram. It means one must be able to state what is to be expressed, using these classes and associations.

The class diagram specification is often considered the most important part in the modeling of an information system.

An object diagram provides examples to illustrate a class diagram.

In particular, it is possible to illustrate a class diagram by showing the corresponding object diagram in the same drawing.

The Class Diagram: summary

A class diagram includes:

- Classes, which represent the basic concepts (client, account, product, etc.).
- Associations, which define the relationships between the different classes.
- Attributes, which describe the characteristics of classes and, in certain cases, of associations.
- Operations, which can be executed on objects of the class.
 - *Operations are not taken into account by **HOPEX Information Architecture** tools (synchronization, generation etc.).*

The class diagram also contains multiplicity definitions.

See the glossary at the end of this guide for definitions of these and other concepts.

Creating a Class Diagram

A class diagram is created from a package.

To create a class diagram:

- > Click the icon of the package concerned and select **New > Class Diagram**.
The diagram opens in the edit window.

A class diagram can describe a package, a use case, a class, or an instance.

CLASSES

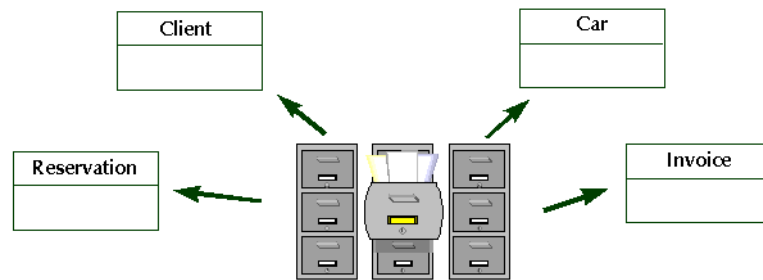
- 6 Definition: Class
- 6 Creating a Class
- 6 Class Properties
- 6 Class Stereotype

Definition: Class

A **class** is described by a list of attributes and operations.

A class is linked to other classes via **associations**. The set of classes and associations forms the core of a class diagram.

We can illustrate the class concept by comparing classes to index cards filed in drawers.



Classes can be technical objects used for programming.

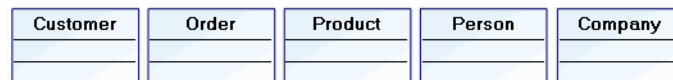
Examples: dialog box, rectangle, string, table, etc.

Classes can represent technical objects used in industry.

Examples: Alarm, Sensor, Zone

Classes can also represent business objects:

Examples: customer, order, product, person, company, etc.



A class can also express a process, such as "Confirm client request", or implement a business rule, such as "Consistency in cost accounts".

- See the glossary at the end of this guide for definitions of these and other concepts.

Creating a Class

To create a class:

1. In the class diagram, click **Class**  in the insert toolbar.
2. Click in the diagram.
The **Add Class** dialog box opens.
3. Enter the **Name** of the class and click the **Add** button.
The class is then placed in the diagram.
 - *In the examples presented in this guide, object names may include spaces, upper case characters, and accented characters. It is important to note that if you have a generation tool using specifications created with **HOPEX UML**, and this tool is more restricted in authorized characters and name lengths, it is preferable to adhere to the more restrictive rules of the **HOPEX UML** specification.*

You can create several classes successively without needing to click on the toolbar each time. To do so, double click the **Class** button.

To return to normal mode, click the  arrow.

You can use the complete name of a class throughout by adding the name of the package to which it belongs to its name, separated by two colons.

Example:

Enterprise::Sales Management::Client.

If one of the packages in the name does not exist, it is automatically created and linked to the class.

Finding an existing class

To find an existing class:

1. In the **Add Class** dialog box, select **List** in the drop-down list box using the arrow.
The list of classes appears.
2. Select the desired class and click **OK**.
The name of the selected class appears in the Add UML Class dialog box
3. Click **Add**.
The class then appears in the drawing.

Class Properties

The properties displayed depend on the class stereotype.

To open the Properties dialog box of a class:

- > Select the class concerned and click the associated **Properties** button in the edit window.
It contains several pages where you can define the class properties.

characteristics page

The **Characteristics** page is used to enter different characteristics of the class:

- Its **Name**, which you can modify.
M You can also modify the name of a class by clicking directly on the name in the drawing.
- The owner of the class (for example, the package).
- The **Visibility** of the class as related to its package:
 - "Public": the class is visible to any element outside the package. This is the default visibility.
 - "Protected": the class is visible to elements that inherit it or have import dependencies with it, and to friends.
 - "Private": the class is only visible to elements that have import dependencies with it and to friends.
 - "Not specified".
) Friends of a class are the classes that are authorized to access its internals. It is possible to specify the friends of a class in the complements tab of the properties dialog box of the class.
- Its **Stereotype** : see [Class Stereotype](#).
- **Comment**: Comments can be used to add key information to diagrams when certain details cannot be displayed in the drawing. These comments are included in the document describing the class diagram.

The other characteristics you can specify are the abstraction, persistence, and activity:

- If the class is **Abstract**, it has no instances. It is only used to group operations or attributes common to its subclasses.
- **Persistence** specifies whether the objects in the class need to exist after the process or thread that created them, or whether they only last as long as the processing.
- Instances of a class which **Is Active** are able to trigger control flows without user intervention.

Example: An instance of the printer class can send an "Out of paper" message to the network administrator screen.

- An **IsRoot** class is a class that has no superclasses in the tree of class generalizations.
- An **IsLeaf** class is a class that has no subclasses in the tree of class generalizations.

You can also specify the Parameters of a parameterized class (for C++).

- See [To specify a parameterized class](#): for further information.

Other properties pages

Other pages allow you to define or view:

- Attributes of a class (see [Attributes](#))
- Operations of a class (see [Operations](#))
- Associated classes (see [Associations](#))
- Instances of a class (see [Object Diagram](#))
- Redefined elements

Class Stereotype

A stereotype is a type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on existing types or classes whose structure they use. Other stereotypes can be created by the user.

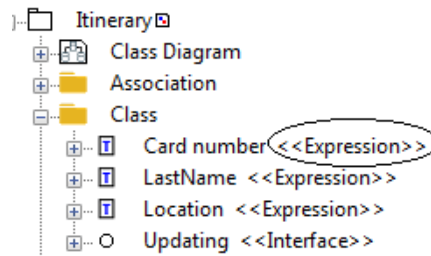
Stereotypes available for a class are:

- **Org-Unit**: represents the role played by something or someone within the enterprise environment of the modeled system.
- **Auxiliary**: class that supports another central or fundamental class, generally by implementing a secondary logic or a control flow.
- **Implementation class**: is used to characterize the classes needed for physical implementation of the system.
- **Metaclass**: class of which the instances are themselves classes. As a general rule, metaclasses are used to build metamodels.
- **Control**: is used for classes that perform processing internal to the system. These generally require contributions from several classes.
- **Entity**: enables description of classes that are passive; that is that do not initiate interactions on their own. They can participate in several use cases and generally outlive any single interaction. They represent objects shared between the different actors that handle them.
- **Enumeration**: datatype containing a list of tabulated values.
- **Expression**: expressions of complex datatypes based on types.
- **Focus**: class that defines the main logic or control flow for the auxiliary class(es) that support it.
- **Boundary**: used to describe classes that are in direct contact with the system environment. Man-machine interfaces are of this type.
- **Interface**: an interface is a named set of operations that describe the behavior of an element. In particular, an interface represents the visible part of a class or package in a contractual client-supplier type relationship.
 - *These are interfaces between the different components of the computer system. These are not interfaces with system users, as those are considered boundary stereotypes. See [Specifying Interfaces](#) for further information.*
- **Worker**: represents a human actor who interacts with the system. A worker interacts with other workers and manipulates entities while participating in use case realizations.
- **Case worker**: a case worker interacts directly with actors outside the system.
- **Internal worker**: an internal worker interacts with other workers and other entities within the system.
- **PowerType**: metatype of which instances are sub-types of another type.
- **Structure**: class that describes a structure used in the programs.
- **Thread**: stereotype used in implementation of an active object as a light business process.
- **Primitive Type**: used to describe the datatypes.

- **Utility:** a class of this stereotype groups global variables and procedures useful for programming, and described as attributes and operations of this class.
- **Schema group:** class describing a type of XML element, the sub-elements of which form a group.
- **XML Document Definition Root:** class that describes the structure of a message exchanged between two systems using the XML language syntax.

Stereotype display option

An option allows you to display stereotypes in the navigation window of objects.



To activate this option:

1. 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

- 6 Definition: Attribute
- 6 Specifying Class Attributes
- 6 Attribute Properties

Definition: Attribute

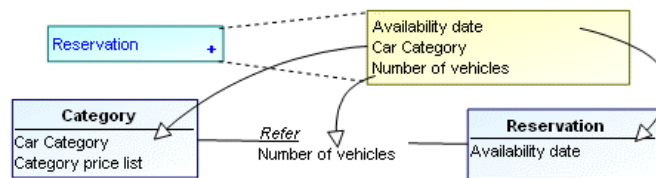
An attribute is a named property of a class. This is the most basic data saved in the enterprise information system.

Examples:

"Client Name" (attribute of the client class).

"Client No." (identifier of the client class).

"Account balance" (attribute of the account class).

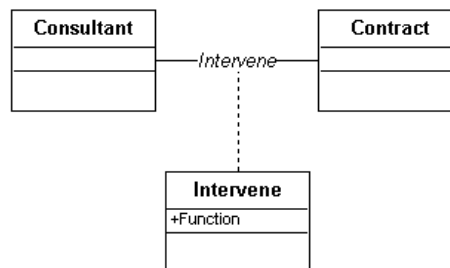


Classes and association classes may be characterized by attributes.

These attributes can be found by studying the content of messages circulating within the enterprise.

An attribute characterizes an association when its value depends on all the classes participating in this association.

In the diagram below, the "Role" that a "Consultant" plays in a "Contract" depends on the consultant and on the contract, and therefore on the "Intervene" association.



Specifying Class Attributes

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

- 6 Definition of an Operation
- 6 Specifying Class Operations
- 6 Operation Properties
- 6 Operation or Signal Signatures
- 6 Operation Parameters
- 6 Operation Methods (opaque behavior)
- 6 Object Diagram
- 6 Operation Exceptions
- 6 Displaying Class Attributes and Operations

Definition of an Operation

An operation is a service that can be requested from an object to affect a defined behavior. An operation has a signature, which may be used to specify the parameters it requires.

Examples:

"Age Calculation" (operation of the client class).

"Print" (operation of the drawing class).

"Calculate due dates" (operation of the loan class).

- *Operations are not taken into account by **HOPEX Information Architecture** tools (synchronization, generation etc.).*

Specifying Class Operations

To specify class operations:

1. Select the class concerned and display its properties.
2. In the **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:typeexpression2,...):returnexpressiontype
```

Names containing spaces or special characters must be enclosed in single quotes ('Client name'). When a name contains an apostrophe, the apostrophe must be typed twice: 'Buyer's Name'

Examples of signatures:

```
Unstock (Product0: Integer(3), Quantity0: Integer): Boolean
'Create order' ('Client name' : Client): Byref Variable
```

In a signature specification, it is possible to specify the package to which a class belongs, followed by two colons.

```
Example: Enterprise::'Sales Management':Client.
```

The listed class is linked to the parameter or return type. If it does not exist, it is created. Any packages listed in the path that do not exist are also created, and linked to the class.

If the package is not specified, a dialog box will enable you to select from similarly named classes.

Operation Parameters

A parameter is the specification of a variable, which can be modified, sent or returned. A parameter can specify a name, a type and a direction. Parameters are used for operations, messages and events.

An argument is a specific value corresponding to a parameter.

In the Properties dialog box of an operation, the **Parameters** section allows you to specify:

- The operation **ExpressionType**, eg. Integer(5).
- Its **defaultValue**, eg. 0.
- Its **Direction**: at input and/or output of the operation.

To create a *parameter* on an operation:

1. Open the operation properties.
2. Select the **Characteristics** page.
3. In the **Parameters** section, click **New**.
The dialog box for creating a parameter opens.
4. Enter the name of the parameter and click **OK**.

Operation Methods (opaque behavior)

A method - or opaque behavior - is a textual representation of implementation of an operation, class or component. It specifies the algorithm or procedure that produces results of an operation or behavior of an element.

To define the method that implements an operation:

1. Open the **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.

M You can hide or show the compartment containing the class attributes in the drawing, by selecting or clearing the "Display of" check box.

Proceed in the same manner to indicate how operations are to be displayed, but instead, select **Operations** in the tree.

SIGNALS

Defining a Signal

A signal is an event that can be explicitly invoked. A signal can have parameters. A signal can be sent to an object or set of objects. It can be invoked as part of the participation of an actor in a use case.

A message can be sent or received by a class. It can also be sent by an operation after an exception.

Specifying Class Signals

Creating a sent or received signal

To specify what signals can be sent or received by a class:

1. Select the class concerned and display its properties.
2. In the properties window, click the drop-down list and select **Complements**.
3. In the menu tree presented, select **sentSignal** or **receivedSignal** then click **Add**.
4. Indicate the name of the signal and click **OK**.

Signal Properties

To open the properties dialog box of a signal:

- > In the properties window for a class, in the **Complements** page, select the signal and click **Properties**.

- The  button displays the hidden commands.

The Properties dialog box of the signal appears.

You can indicate for a signal:

- Its **Stereotype** to specify its use:
 - **Exception**: an error signal is generated when an exception occurs during the execution of an operation.
- Its **Visibility** related to the package:
 - **Public**: this is the default visibility. The signal is visible to any element outside the package.
 - **Protected**: the signal is visible to inherited elements or friends.
 - **Private**: the signal is visible to its class or to its friends.
 -) *Friends of a class are the classes that are authorized to access its internals. It is possible to specify the friends of a class in the complements tab of the properties dialog box of the class.*
- The **ExpressionType** of the signal (see *expression type*)..
 -) *The ExpressionType of a signal specifies the type of variable returned by the signal on its receipt by the addressee.*

A signal can be a request to **Vote** sent to each active object, asking if it is possible to perform a specific action such as closing a Windows session.

A signal can be a general **Broadcast** to all active objects.

Signal parameters

The *Parameters* of the signal are specified in the **Parameters** tab of its Properties dialog box. You can specify:

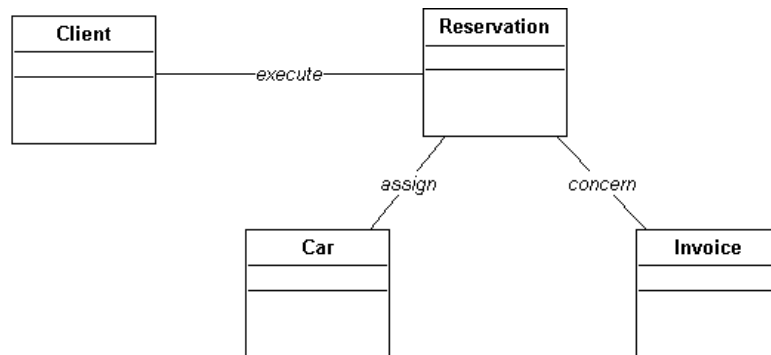
- The operation **ExpressionType**, eg. Example: Integer(5).
- Its **Default Value**. Example: 0.
- Its **Direction**: at input and/or output of the operation.
 -) *A parameter is the specification of a variable, which can be modified, sent or returned. A parameter can specify a name, a type and a direction. Parameters are used for operations, messages and events.*
 -) *An argument is a specific value corresponding to a parameter.*

ASSOCIATIONS

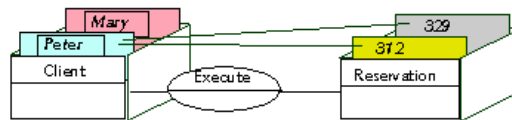
An association is a relationship existing between two classes.

An association is binary when it links two classes, ternary when it links three classes, etc.

Associations can be compared to links between index cards.



The following drawing provides a three-dimensional view of the situations a class diagram can store.



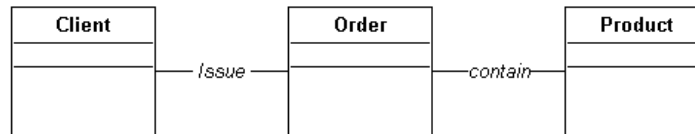
Peter and Mary are clients. Peter has made reservations numbers 312 and 329.

A class diagram should be able to store all situations in the context of the company.

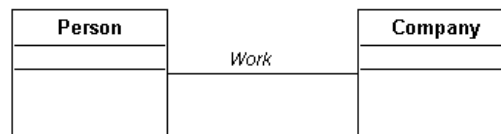
- The diagram should not allow representing unrealistic or aberrant situations.

Examples of associations:

- A client issues an order.
- An order includes several products.



- A person *works* for a company.



- An alarm is triggered by a sensor.

A sensor covers a zone.

A window displays a string of characters.

Creating an Association

To create an *association*:

1. In the class diagram, click **Association**  in the objects toolbar.
2. Click one of the classes concerned and drag the mouse to the other class before releasing the button.
The Creation of Extension dialog box appears.
3. Enter the name of the association to be created.
 - You can also select an existing association.
4. Click **Add**.
The association is indicated by a line in the diagram.
 - If you make a mistake, you can delete an element or a link by right-clicking it and selecting the Delete command in the pop-up menu.

Roles (or Association Ends)

It is possible to describe the different *roles* played by the classes in associations and to specify their multiplicity and their navigability.

Each end of an association specifies the role played by the class in the association.

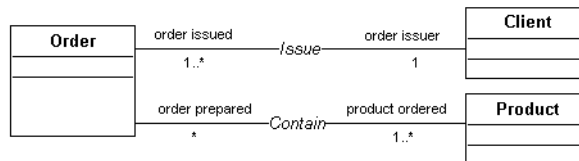
The role name is distinguished from the association name in the drawing by its position at the link end. In addition, the role name appears in a normal font, while the association name is italicized.



When two classes are linked by only one association, the name of the classes is often sufficient to describe the role. Role names are useful when several associations link the same two classes.

Examples of roles:

- A client is the *order issuer*.
- An order is *issued* by a client.
- An order is *prepared* from products.
- A product is *ordered*.



A person is an *employee* of a company.

A company is the *employer* of these persons.

An alarm is *triggered* by one or more sensors.

A zone is *covered* by a sensor.

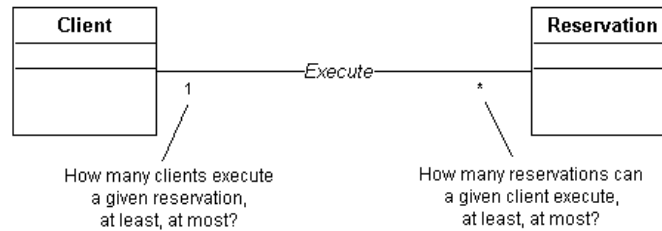
One or more strings are *displayed* in a window.

Multiplicity of a Role

) *Multiplicity specifies the interval between minimum and maximum values of cardinalities for a set. This is primarily indicated for each role that classes play in an association. It can assume the values *, 0..1, 1, 1..*, 2..*, 4..10, etc. The default value is *.*

) *Cardinality is the number of elements contained in a set.*

The *multiplicity* expresses the minimum and maximum number of instances of a class that can be linked by the association to each instance of the other class.

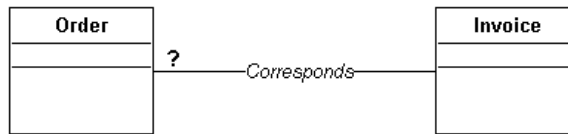


The usual multiplicities are "1", "0..1", "*" or "0..*", "1..*", and "M..N" where "M" and "N" are integers:

- The "1" multiplicity indicates that one and only one instance of the class is linked by this association to each instance of the other class.
- The "0..1" multiplicity indicates that at most one instance of the class can be linked by this association to each instance of the other class.
- The "*" or "0..*" multiplicity indicates that any number of instances of the class can be linked by the association to each instance of the other class.
- The "1..*" multiplicity indicates that at least one instance of the class is linked by the association to each instance of the other class.
- The "M..N" multiplicity indicates that at least M instances and at most N instances of the class are linked by the association to each instance of the other class.

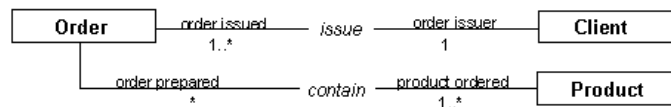
1	One and one only
0 / 1	Zero or one
M..N	From M to N (natural integer)
*	From zero to several
0..*	From zero to several
1..*	From one to several

The following example illustrates the significance of the different multiplicities:



- 0..1 : An order corresponds to zero or at most one invoice.
- * : No restriction is placed on the number of invoices corresponding to an order.
- 1: Each order has one and only one corresponding invoice.
- 1..* : Each order has one or more corresponding invoices.

Other examples of multiplicity:



- 1..* : A client can issue one or more orders.
- 1: An order is issued by one and only one client.
- 1..* : An order contains one or more products.
- * : A product can be contained in any number of orders, including no orders.
- 0..1 : A person works for a company.
- 1..* : An alarm is triggered by one or more sensors.
- 1: A sensor covers one and only one zone.
- 1..* : A window displays one or more strings.

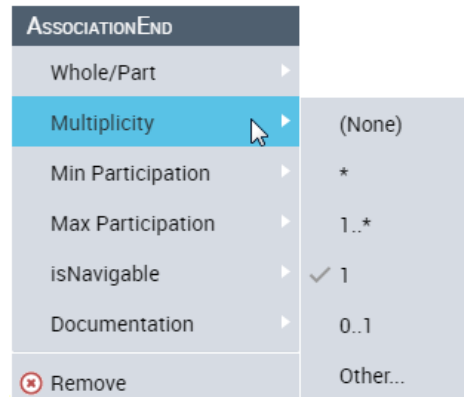
Specifying role multiplicity

To specify association end multiplicity:

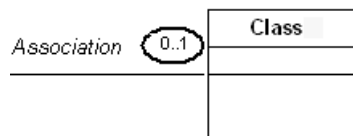
1. Right click on the part of the line of the association that is located closest to the class.

2. Select **Multiplicity** then the desired value.

- If the menu you see does not propose multiplicity, check that you clicked on that part of the line indicating the role and not the association.



The multiplicity now appears on the role.



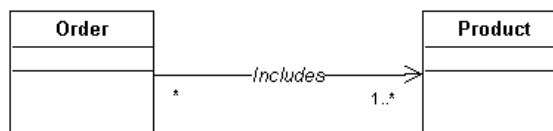
M All the information specified with the pop-up menu can also be viewed and modified in the role properties dialog box.

Association End Navigability

IsNavigable specifies in which direction(s) an association between two classes can be traversed. To avoid crowding the drawing, this is only indicated when only one direction is possible.

Example of navigability:

- It is important to be able to find out what products are contained in an order.
- However, it is rarely useful to be able to find all orders that concern a product.



Specifying navigability for a role

To indicate that an association is navigable in one direction only:

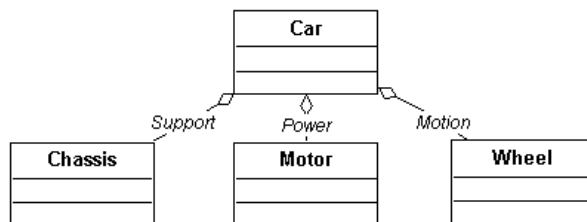
1. Right-click the non-navigable role.
2. Select **IsNavigable > No**.

An arrow representing the navigability now appears for the other role.

Association End Aggregation

Aggregation is a special form of association, indicating that one of the classes contains the other.

Example: A car includes a chassis, an engine, and wheels.



Specifying role aggregation

To specify role aggregation:

1. Right-click the role.
2. Select **Whole/Part > Aggregate**.

- If the menu you see does not propose aggregation, check that you clicked on that part of the line indicating the role and not the association.

A diamond now appears on the role, representing the aggregation.

Association End Composition

A composition is a strong aggregation where the lifetime of the components coincides with that of the composite. A composition is a fixed aggregation with a multiplicity of 1.

Example: An order consists of several order lines that will no longer exist if the order is deleted.

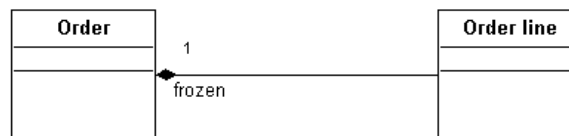
Composition is indicated by a black diamond.



Role Changeability

Read Only specifies whether the role played by a class in an association may be modified after it has been created. By default, the role of a class in an association is considered changeable.

Example: An order includes an order line for each of the ordered products. These order lines can no longer be modified after the order has been saved.



You can indicate whether a role is changeable using the role pop-up menu or the role properties dialog box.

The **Read Only** characteristic of the role can have the following values:

- **Add only**: it is still possible to link new objects with this association, but already linked objects cannot be unlinked.
- **Read Only**: linked instances can no longer be unlinked. Nor is it possible to add a new link.
- **No Restriction**: new instances can be linked or unlinked at any time with no constraints.

Role Order

It is possible to specify whether or not a role is Ordered. For example, for a client order, it can be useful to store the sequence in which its lines appear.

To specify that a role is ordered:

1. Open the **Properties** dialog box of the role.
2. In the **Characteristics** page, select the **IsOrdered** check box.

Role Static Property

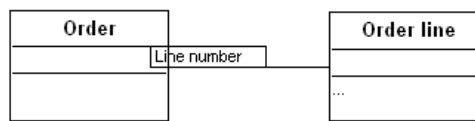
As for an attribute, it is possible to specify if a role can take specific values for each class instance, or take a value characterizing the entire class:

1. Open the properties dialog box of the role.
2. Click the **Characteristics** page.
3. In the **Static** box, select:
 - "Yes": so that the role can take a value characterizing the entire class.
 - "No": so that the role can take a different value for each class instance.

Role Qualifier

A qualifier is an attribute whose values partition the set of objects related to an object across an association.

Example: An order includes several order lines. The order line number can be used as the qualifier that identifies each line.



To define a qualifier:

1. Right-click the role and select **Properties**.
The Properties dialog box of the role opens.
2. Select the **Qualifiers** page.
3. To add a new qualifier to the role, click **Add**.
4. Enter the name of the qualifier.
5. Click **Add**.

Several qualifiers may be needed to uniquely identify each object in a class.

For example, each square on a chessboard is identified by its row number and column number on the chessboard.



Overloading a Role

A role can inherit a role defined at higher level. Overloading enables definition of additional properties on an inherited role.

To overload a role:

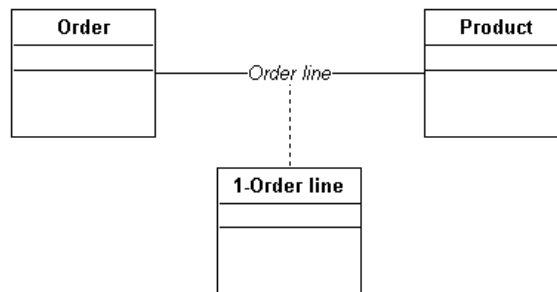
1. Open the properties dialog box of the role.
2. In the properties window, click the drop-down list and select **Characteristics**.
3. In the **Roles** section, select **Overloaded Role**.
4. Click **Add**.
The Query dialog box appears:
5. Search and select the role in question.
6. Click **OK**.

Association Classes

An association class is an association that also has class properties as attributes.

It is helpful to create an association class in order to specify the characteristics of an association.

For example, the quantity of the requested product needs to be specified on each order line.



To create an association class:

1. Create a new class.
2. Using the **Link** button, create a link between the class and the association.

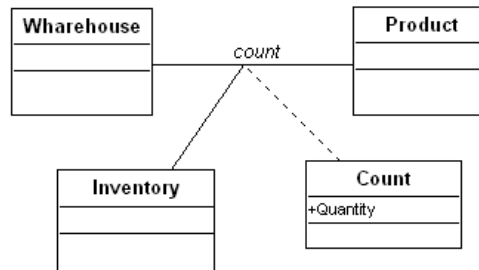
The association class is linked to the association by a dotted line.

- As for standard classes, it is possible to hide the compartments and resize the association class using the *Display* command in its pop-up menu.

Displaying an N-ary Association

Certain associations associate more than two classes. These associations are generally rare.

Example: When taking inventory, a certain quantity of product was counted in each warehouse.

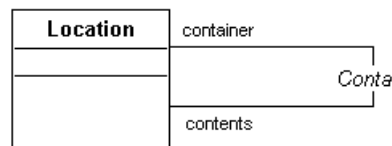


To create a ternary association:

1. First create the association between the two classes.
2. Click the **Association Role** button .
3. Draw a link between the association and the third class.

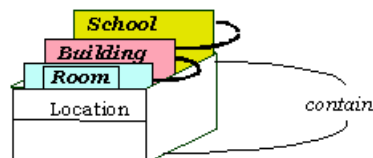
You can then proceed as described above to create an association class if needed.

Reflexive Associations



Certain associations use the same class several times.

A classroom, a building, and a school are all locations.



A classroom is contained in a building, which is contained in a school.

A reflexive association concerns the same class at each end.

Creating a reflexive association

To create a reflexive association:

1. Click the **Association** button  in the toolbar:
 2. Click on the class concerned and drag the mouse outside the class, then return to it and release the mouse button.
- The reflexive association appears in the form of a half-circle.

- *If there is an association of a class to itself, the roles need to be named in order to distinguish between the corresponding links in the drawing.*

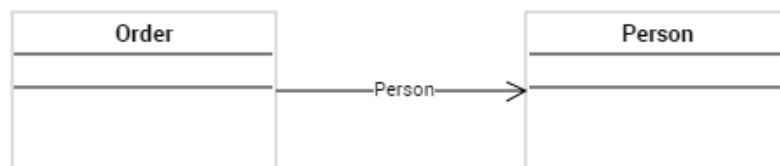
THE PARTS

In a class diagram, a part represents a role played by an instance of a class or component at execution of a task.

A part belongs to a class. Ownership is specified on the link of the part.

In the example below, the "Order" class comprises the "Person" class.

The part is owned by the "Order" class and references the "Person" class.



Creating a Part between two Classes

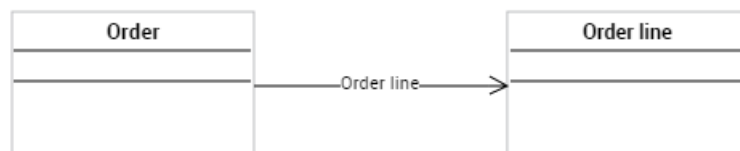
A part is a directional link that connects two classes only.

To build an part between two classes:

1. In the objects toolbar of the class diagram, click **Part**.
2. Draw a link from the owner class to the referenced class.
The name of the part is automatically defined.

Defining the Identifier of a Class via a Part

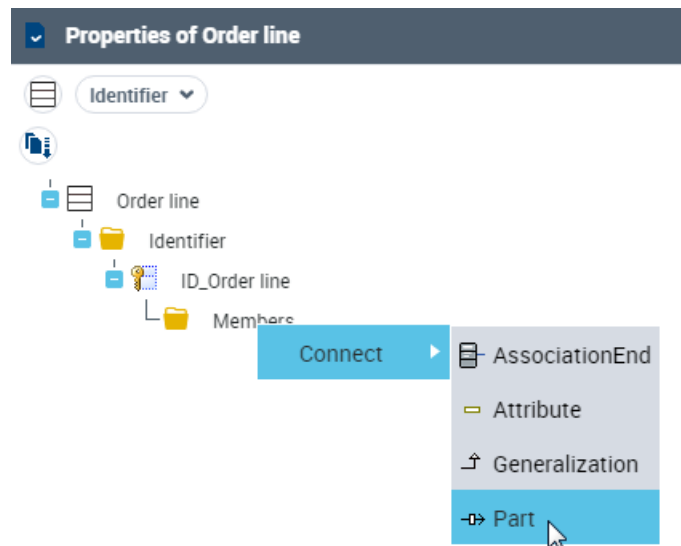
In the example below, the identifier of the "Order line" class can be defined from the "Order" class through the "Order line" part.



To define the identifier of the "Order line" class:

1. Display the properties of the "Order line" class.
2. Select the **Identifier** page.

3. Right-click the **Members** folder and select **Connect > Part**.



4. Select the proposed part.
5. Click **OK**.

Multiplicities of the Associated Classes

With multiplicities you can specify the minimum and maximum number of instances linked by the part.

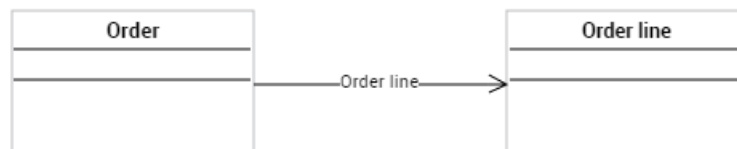
Example: 1 order comprises 1 or several order line(s).

Multiplicity of the class referenced by the part

The multiplicity of the referenced class must be indicated on the part link.

To define the multiplicity of the referenced class:

1. Right-click the part link.

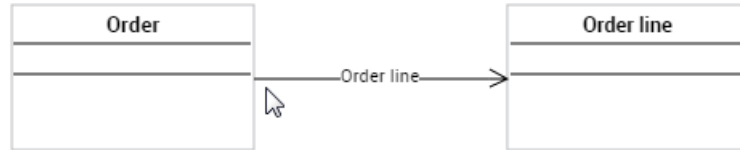


2. Select **Multiplicity** then the desired value.

Multiplicity of the owner class of the part

To define the multiplicity of the owner class of the part:

1. Right-click the part role associated with the owner class.



2. In the pop-up menu that appears, select **Multiplicity** then the desired value.

Aggregation and Composition Relationships

On the part that links two classes, you can define an aggregation or composition relationship.

) *Aggregation is a special form of association, indicating that one of the entities contains the other.*

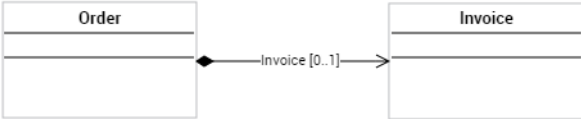
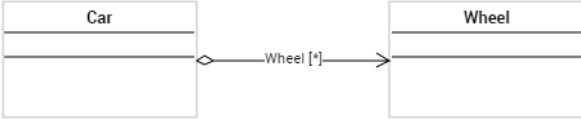
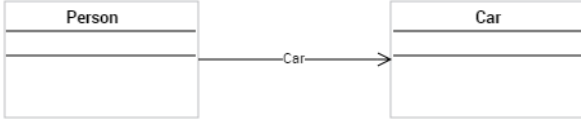
) *A composition is a strong aggregation where the lifetime of the components coincides with that of the composite. A composition is a fixed aggregation with a multiplicity of 1.*

To define a composition or an aggregation link between two classes:

1. Right-click the part.
2. Select **Whole/Part** then the desired value:
 - **Aggregate**
 - **Composite**

Associated multiplicities

The following table presents the multiplicities automatically associated with aggregations and compositions.

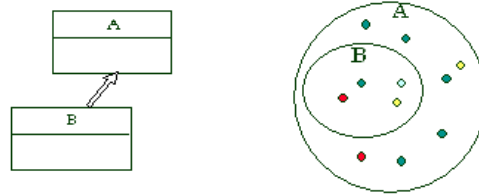
		Corre- sponding multiplicity	Example
Composition		1	 1, 1..*
Aggregation		0 / 1	 0..1, 1..*
None		*	 *, *

GENERALIZATIONS

A generalization represents an inheritance relationship between a general class and a more specific class. The more specific class is fully consistent with the more general class and inherits its characteristics and behavior. However, it also contains additional information. Any instance of the more specific class is also an instance of the general class.

- 6 [What is a Generalization?](#)
- 6 [Multiple Subclasses](#)
- 6 [Advantages of Subclasses](#)
- 6 [Multiple Inheritance](#)
- 6 [Creating a generalization](#)
- 6 [Discriminator](#)

What is a Generalization?



Class A is a generalization of class B. This implies that all objects in class B are also objects in class A. In other words, B is a subset of A.

B is then the subclass and A the superclass.

Example A: Person, B: Bostonian.

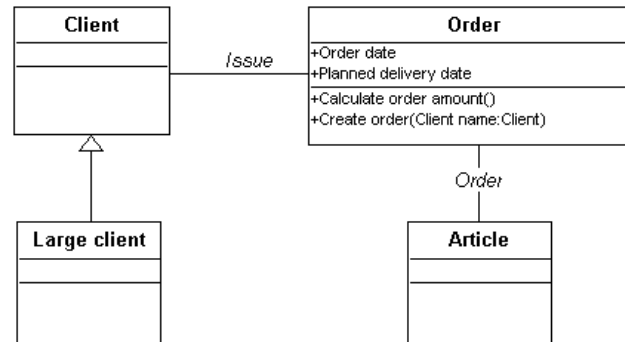
B is a subset of A, so the objects in class B inherit the characteristics of those in class A.

It is therefore unnecessary to redescribe for class B:

- Its attributes
- Its operations
- Its associations

Example

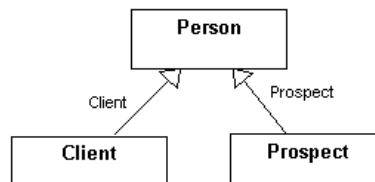
The "Large Client" class, representing Clients with a 12-month revenue exceeding \$1 million, can be a specialization of the Client class (origin).



In the above example, the associations and attributes specified for "Client" are also valid for "Large client".

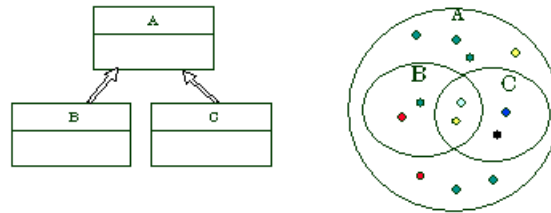
Other examples of generalizations:

- Prospect and client are two subclasses of "person".



- Export order is a subclass of the "order" class.
- Individual person and corporate person are two subclasses of the "person" class.
- Polygon, ellipse, and circle are subclasses of the "shape" class.
- Oak, elm, and birch are subclasses of the "tree" class.
- Motor vehicle, all-terrain vehicle, and amphibious vehicle are subclasses of the "vehicle" class.
- Truck is a subclass of the "motor vehicle" class.

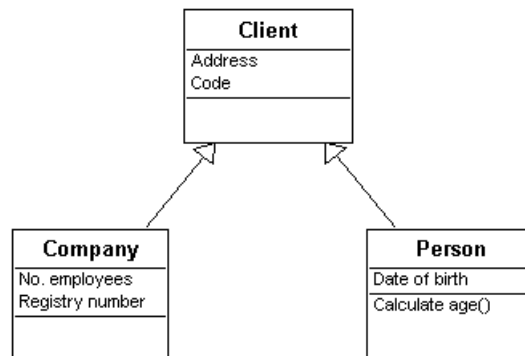
Multiple Subclasses



When a class has multiple subclasses, they:

- are not necessarily exclusive.
- do not necessarily partition the set.

Advantages of Subclasses

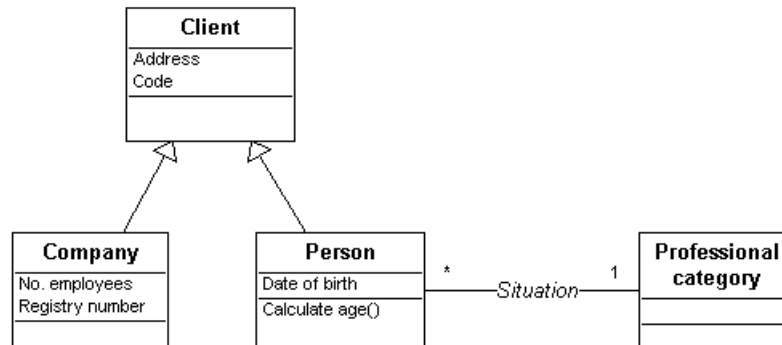


A subclass inherits all the attributes, operations, and associations of its superclass, but can have its own attributes or associations that the superclass does not have.

A subclass can also have specific attributes. These only have meaning for that particular subclass. In the above example:

- "Registry number" and "number of employees" only have meaning for a "company".
- "Date of birth" is a characteristic of a "person", not a "company".
- It is also useful to calculate the "age" of a "person". This attribute and this operation are generally not needed for a "company".

A subclass can also have specific *associations*.



- A "person" falls into a "socio-professional group": "manager", "employee", "shop keeper", "grower", etc. This classification makes no sense for a "company". There is also a classification for companies, but it differs from that for persons.

Multiple Inheritance

It is sometimes useful to specify that a class has several superclasses. The subclass inherits all the characteristics of both superclasses. This possibility should be used carefully.

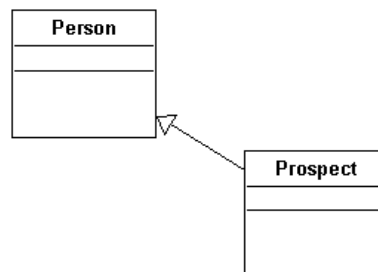
- *Multiple inheritance is not taken into account when generating tables.*

Creating a generalization

To create a generalization:

1. Click the **Generalization** button  in the toolbar.
2. Click the subclass concerned, and drag the mouse to the superclass before releasing the button.

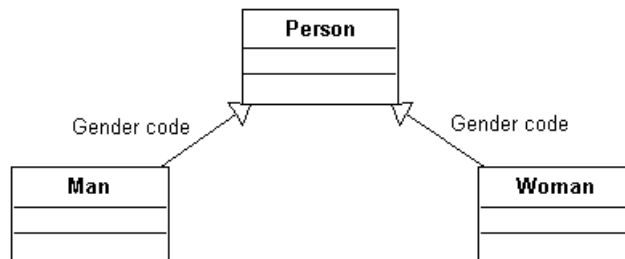
The generalization is now indicated in the diagram by an arrow.



Discriminator

The discriminator is the attribute of a generalization, the value of which distributes objects into the sub-classes associated with the generalization.

For example, the gender code attribute divides the objects in the person class into the man and woman subclasses.



You can define discriminator(s) in the generalization properties dialog box, under the **Discriminators** page.

- It is also possible to specify whether a generalization:

Is Disjoint: An instance cannot belong to two subclasses of the generalization simultaneously.

Is Complete: All instances of the superclass belong to at least one of the subclasses of this generalization.

SPECIFYING INTERFACES

An interface represents the visible part of a class or package in a contractual client-supplier type relationship. The interface is a class stereotype.

An interface is a named set of operations that describe the behavior of an element. In particular, an interface represents the visible part of a class or package in a contractual client-supplier type relationship.

These are interfaces between the different components of the computer system.

Creating an Interface

To create an interface class in a class diagram:

1. In the toolbar, select **Interface** .
2. Click in the diagram.
3. In the dialog box that appears, enter the name of the interface and click the **Add** button.
The interface class then appears in the diagram.

You can then specify the operations of the interface as for any other class.

Connecting an interface to a class

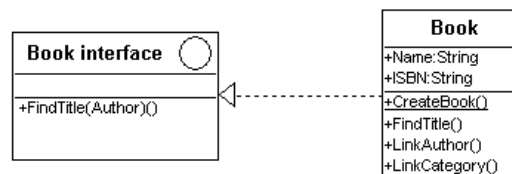
When you connect a class to an interface, you must specify if it is an interface required or provided by the class.

A required interface is an interface necessary for object operation.

A provided interface is an interface made available by an object to other objects.

To connect an interface to a class:

1. Click the **Link** button .
2. Create the link from the class to the interface.
A dialog box appears:
3. Indicate the type of link to be created: provided interface or required interface.
- Other types of links, specific to classes, can be displayed.
4. Click **OK**.



SPECIFYING DEPENDENCIES

In the class diagram, to indicate that a package references a class or another package:

1. Click the **Link** button 
2. Then carry out the link from a package to the package or class that it references.

- The **Views**  button allows you to specify the buttons that you want to appear in the objects toolbar.

SPECIFYING PARAMETERIZED CLASSES

A parameterized class enables definition of characteristics and a behavior that varies as a function of the value of certain parameters. For example, a parameterized class can be used to manage object lists. In this case the parameter will be the object type to be managed in the form of a list. This type of class is implemented in particular in C++ language.

To specify a parameterized class:

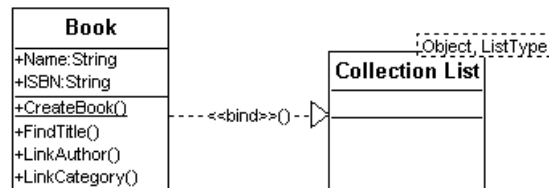
1. Open the **Characteristics** property page of the class.
2. You can enter the parameters and specify their type if necessary.

TemplateParameters: Object, ListType

The class parameters are displayed at top right.

To link a class to a parameterized class:

1. Click the **Link** button .
2. Create the link from the class to the parameterized class.



CONSTRAINTS

A constraint is a declaration that establishes a restriction or business rule generally involving several classes.

Most constraints involve associations between classes.

Examples of constraints:

- The person in charge of a department must belong to the department.
- Any invoiced order must already have been delivered.
- The delivery date must be later than the order date.

A sensor covering a zone can trigger an alarm for that zone only.

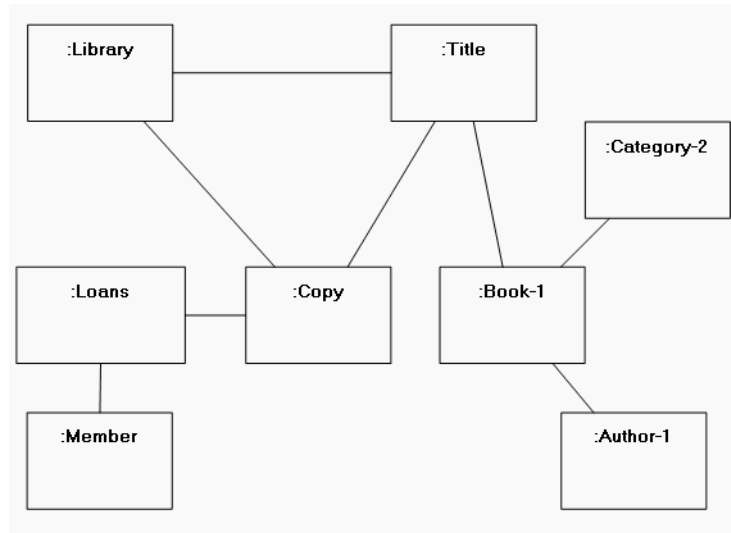
To create a constraint in the class diagram:

1. Click the **Constraint** button  in the object toolbar.
 - If it is not displayed, select **View Views and Details** and select the "Constraints" check box.
2. Then click one of the associations concerned by the constraint, and drag the mouse to the second association before releasing the mouse button. The Add Constraint dialog box appears.
3. Enter the name of the constraint, then click **Add**. The constraint then appears in the drawing.
 - You can link a constraint to other classes or associations using the **Link** button .

OBJECT DIAGRAM

An object diagram or instance diagram contains objects with values illustrating their attributes and links. It shows in detail the state of the system at a given moment.

You can create the object diagram of a class, component, package or use case.



Objects

An object is an entity with a well-defined boundary and identity that encapsulates state and behavior. Its state is represented by the values of its attributes and its relationships with other objects. Its behavior is represented by its operations and methods. An object is an instance of a class.

Examples of objects:

- Business objects:
 - John Williams, Elizabeth Davis, Paul Smith are instances of the person class.
 - Orders 10533 and 7322 are instances of the order class.
 - Sony SPD-1730 Monitor, Compaq Deskpro 200 are instances of the item class.
 - Dupont and Burger King are instances of the company class.
 - Technical objects used for programming:
 - Dlg_Order_Create, Dlg_Client_Query are instances of the window class.
 - Str_Client_Name, Str_Product_Comment are instances of the string class.
- *The objects represented in an object diagram can be instances of a class, package, use case, component, or node, to enable defining sequence diagrams at the desired level of detail.*

Creating an object (instance)

To create an object:

1. Click the **Instance** button.  |
You can create objects of different types. The arrow at the right of the button offers a shortcut to Class and Component object types, the most frequently used.
2. Then click in the diagram work area.
The window for adding an instance opens.
3. Enter the instance **Name**.
4. Specify the **Instance Type** if necessary.
5. Click **Add**.

The instance appears in the diagram.

Instance properties

To open the properties dialog box of an instance:

- > Select the instance in question and click **Properties** in the edit window if it is not activated.

It contains several pages where you can define the properties of an instance.

In the **Characteristics** page, you can:

- Select the **Instance kind** (Actor, Class, etc.).
- You can specify of which **Class, Actor**, etc. this object is an instance.
- Indicate a name for this instance.
- Specify its **Stereotype**.

Value of an attribute

To specify the value of an attribute:

1. Display the properties of the instance of the class that contains the attribute.
2. Select the **Attributes** page.
3. In the corresponding column, indicate the value of the attribute. You can specify an instanced value or a constant value.
 - **Instanced value**: click in this column to display the list of possible instances for the selected attribute. These are variable values.
 - **Value**: click in the column and enter the value of the attribute.

Links

A link represents an instance of an association between two objects.

Examples of links between objects:

- Order no. 10733 was placed by John Williams.
- Order no. 10733 includes the products Sony SPD-1730 Monitor and Compaq Deskpro 200.
- John Williams works for Dupont.
- The window Dlg_Client_Query displays the string Str_Client_Name.

Creating a link

To create a link:

1. Click the **Link** button  in the diagram toolbar.
2. Click one of the objects concerned, and drag the mouse to the second object before releasing the mouse button.

The link then appears in the diagram.

If there is already a link between the two objects, a dialog box asks you to choose an existing link or create a new one.

Link properties

To open the properties dialog box of a link:

- › Select the center of the link to display its **Properties**.
 - *If you do not click on the center of the link, the properties dialog box for one of the roles will be displayed.*

Under the **Characteristics** page, you can specify:

- The **Name** of the link.
- The link **Stereotype**.
- The **Association** corresponding to the link.
- The **Visibility** of the link.
- The **Package** containing the link.

In the **Link Role** page:

- For each **Instance** connected by this link, the name of the **Role** and its **Multiplicity**.
 - *Only the associations between the classes of the two instances are listed.*

Role properties

To open the properties dialog box of a role:

1. In the properties window of a link, select the **Link Role** page.
2. Select the role in question and click **Properties**

- The  button displays the hidden commands.

The Properties dialog box of the role opens.

In this dialog box you can specify:

- A **Name** for the role.
- The **Role** for this instance.
- The **Multiplicity** for the role.
- The values for the role *Qualifiers*, defined at the class level.



STRUCTURE AND DEPLOYMENT DIAGRAMS



In addition to class and object diagrams, structural diagrams include:

- 6 [The Package Diagram](#), enabling organization of elements of the model
- 6 [The Component Diagram](#), highlighting dependency relationships between components
- 6 [Composite Structure Diagram](#), describing interactions between components and their parts

THE PACKAGE DIAGRAM

A package diagram enables organization of modeling elements, in order to partition the work involved in specification and development.

An element should only appear in a single package.

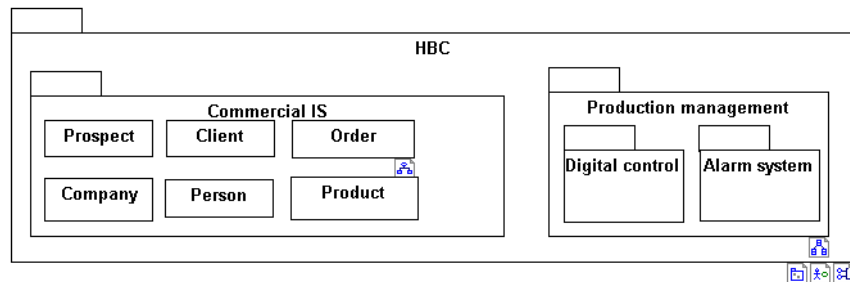
Dividing into packages is generally carried out so as to minimize interactions between different packages.

Example of a package diagram

The "HBC" package contains the "Commercial IS" and "Production Management" packages.

The "Production Management" package can be divided into two packages, "Digital Control" and "Alarm System".

The "Commercial IS" package contains the "Prospect", "Client", "Company", "Person", "Order", and "Product" classes.



Creating a Package Diagram

A package diagram is created from a package.

To create a package diagram with **HOPEX IT Architecture V2** from the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit area window, click the **Packages** tile.
3. Display all packages.
4. Right-click the name of the package concerned and select **New > Package Diagram**.

The diagram opens in the edit window.

Defining Packages

A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.

Examples of *packages*:

- The commercial information system.
- Accounting.
- Production management.
- Digital control of a machine.
- Inventory management.
- Alarm system and telephone management.

To add in the diagram an existing package:

1. In the package diagram, click the **Package** button in the object toolbar, then click the workspace.
2. In the **Add Package** dialog box, select **List** in the drop-down list box using the arrow  .
The list of packages appears:
3. Select the desired package and click **OK**.
The name of the diagram appears in the **Add UML Package** dialog box.
4. Click **Add**.

The package appears in the diagram.

Defining Classes

The package diagram can be used to place classes in different packages.

To quickly add a set of classes to the package diagram:

1. Click **Main Menu > Advanced Query** to open the Query assistant.
2. In the wizard, select the "Class" metaclass and click **Query**  .
The list of repository classes appears.
3. Select the classes you want and drop them in the diagram.

Specifying Dependencies in a Package Diagram

Links allow you to indicate if a package contains or references a class or another package.

To indicate that a package references a class or another package:

1. Click the  button.

2. Then carry out the link from a package to the package or class that it references.
A dialog box asks you the type of link to be created.
3. Select "Referenced package" or "Referenced class" as required.

THE COMPONENT DIAGRAM

A *component diagram* shows the interdependency of software components and *interfaces* (it defines who uses what).

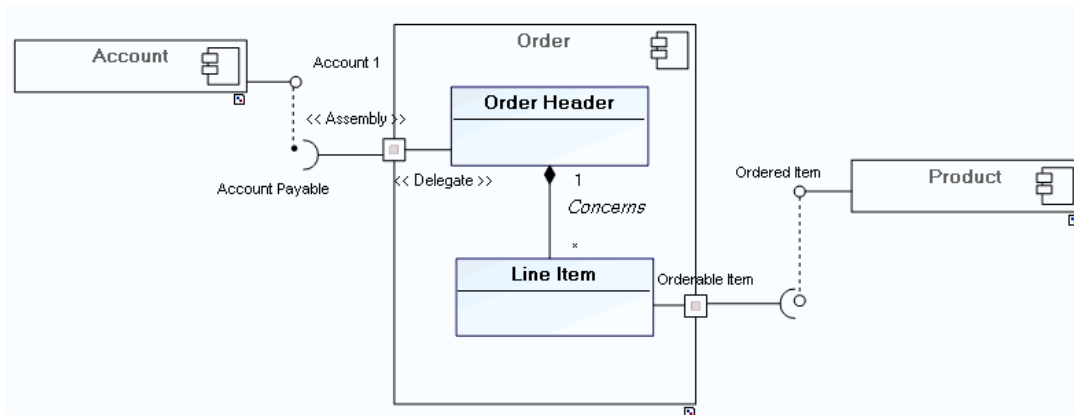
) A component is an implementation element of the system: it can be software, a program, a code element, or a physical element such as a work document.

) An interface represents the visible part of a class or package in a contractual client-supplier type relationship. The interface is a class stereotype.

A component diagram contains components and classes of the "Interface" stereotype. It is also possible to specify packages implemented by the components.

Example of a component diagram

This diagram describes the elements contained in the "Order" component and the interactions of these elements with external components.



Creating a Component Diagram

In **HOPEX IT Architecture V2**, you can create a component diagram using a component or package.

To create a component diagram with **HOPEX IT Architecture V2** from the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit area window, click the **Packages** tile.
3. Display all packages.
4. Right-click the name of the package concerned and select **New > Component Diagram**.
The diagram appears in the edit window.

Components

A component represents a modular part of a system that encapsulates its content, and which can be replaced in its environment. A component defines its behavior by means of interfaces that it provides and requires.

One component can be replaced by another if their interfaces conform.

A component can be a software package, program, code unit, etc.

It is represented by the following icon: 

Interfaces

Creating component interfaces

An interface represents the visible part of a class or package in a contractual client-supplier type relationship.

The interface is a particular type of class.

To create a class of "Interface" stereotype in the composite structure diagram:

1. Click the **Interface** button  , then click in the diagram.
2. In the dialog box that appears, enter the name of the class.
3. Click **Add**.

– You can specify the details for the interface in terms of attributes and operations in the class diagram in the same way as for a class.

Linking interfaces to other objects

Two link types enable differentiation of required interfaces and provided interfaces.

A required interface is an interface necessary for object operation.

Example: the "Purchasing Management" component requires the "Product" interface for its operation to be able to associate a purchase order with products ordered.

A provided interface is an interface made available by an object to other objects.

Example: the "Product Management" component makes available the "Product" interface.

You can define interfaces required and provided by an object independently of other objects.

To specify that an interface is supported or required by an object:

1. Click the **Connect**  button and drag the link from the object to the interface.
A dialog box appears:

2. Indicate the type of link to be created.
 - Required interface
 - Supported interface
3. Click **OK**.

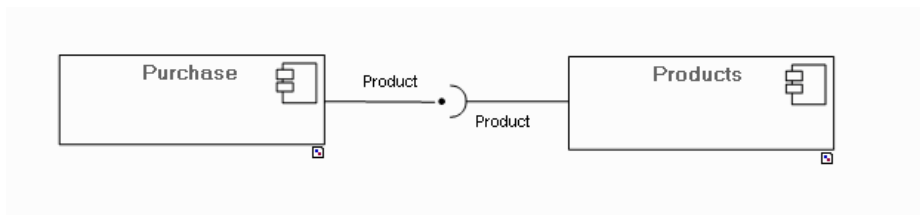
The link then appears in the diagram.

The interface shape differs according to link type:

Connecting interfaces

Two interfaces can be interconnected. This connection is modeled by a connector.

You can also indicate that an interface provided by an object is required by another. Here it is one and the same interface.



Ports

Ports enable connection of a component to its parts or to its environment.

Ports are represented by a square in the diagram, placed at the edge of the described element when they assure connection with the exterior.

They are connected to components by connectors.

Ports can specify queries sent and services provided by the component, as well as queries and services they may require from other parts of the system. These queries and services are represented by classes of interface type.

You can view interfaces associated with a port in the properties dialog box of the port, in the **Provided and Required Interfaces** tab.

Connectors

Connectors enable connection of diagram objects.

Connectors of simple type do not specify a particular connection type, they are notably used to connect instances of objects described in collaborations.

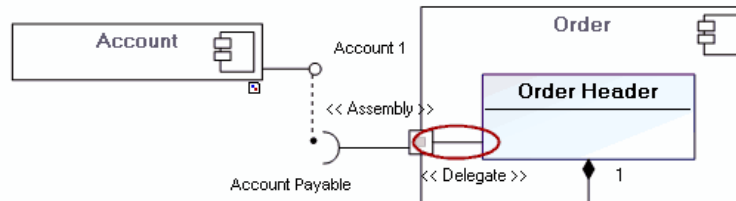
In the composite structure diagram, it is possible to specify the type of connector between two objects: Assembly or Delegate.

Delegate connector

A "Delegate" type connector indicates the redirection of queries to a component element responsible for their execution.

The delegation link can be made directly between the component port and the component element, or between the component port and the element port.

Below, the "Order" component delegates management of accounts to be debited to the "Order Header" class.



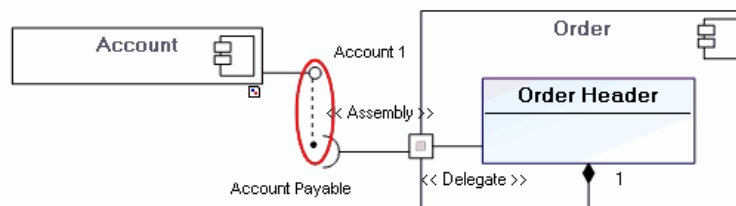
Assembly connector

An "Assembly" type connector is a connector between two or more components or ports indicating that one or more components provide services that others use.

- These can be other objects or components.

To connect ports or components that share an interface, you can also use "Provided Interface" and "Required Interface" links.

An "Assembly" type connector connects the interface provided by the "Account" component to the interface required by the "Order Header" class.



COMPOSITE STRUCTURE DIAGRAM

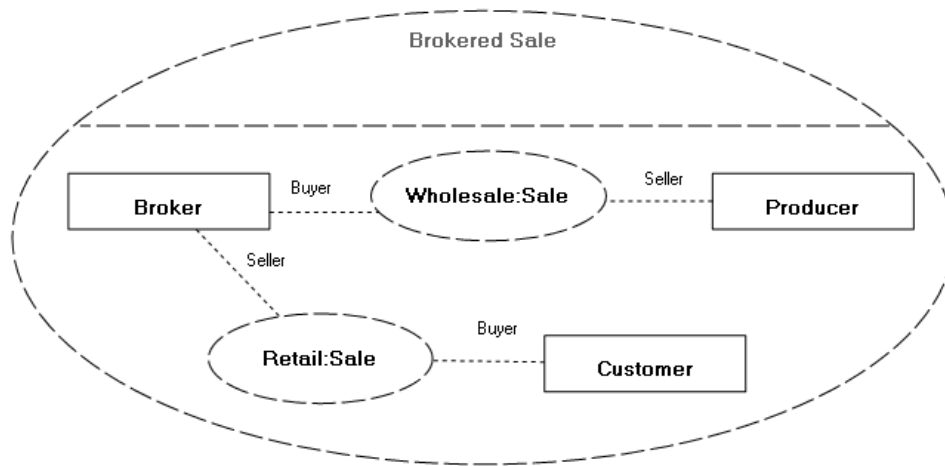
The composite structure diagram enables description of the internal structure of a component, a package or a structured class.

It also enables specification of collaborations that intervene between elements of the structure in execution of a task, highlighting the role played by each element in the collaborations.

Elements of this diagram are *parts*, ports by which parts interact with the exterior, and connectors linking the parts between themselves and with the ports.

Example of a composite structure diagram

This diagram describes the role played by parts in the "Brokered Sale" collaboration.



Creating a Composite Structure Diagram

To create a composite structure diagram with **HOPEX IT Architecture V2** from the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit window, click the **Components** tile.
3. Right-click the name of the component concerned and select **New > Component Diagram**.
The diagram appears in the edit window.

Parts

A part represents a role played by an instance of a class or component at execution of a task.

Parts are interconnected by connectors or dependencies.

A part can also be connected, via a connector, to a port which acts as interface between the described component and the exterior.

For more details on these elements, see:

- 6 ["Connectors", page 291](#)
- 6 ["Dependency links", page 295](#)
- 6 ["Ports", page 291.](#)

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

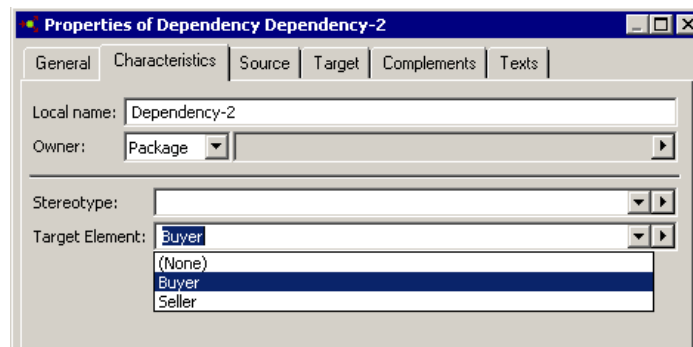
A collaboration use represents application of the structure described by a collaboration to a particular situation implementing classes or specific instances. These classes or instances therefore play roles defined in the collaboration.

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

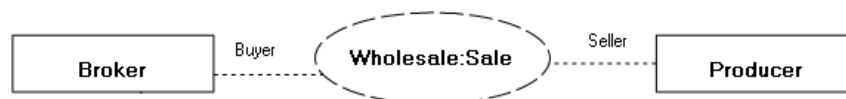
In the case of a purchasing request between two instances of an actor, a collaboration is used. This collaboration connects two roles: the role of buyer and the role of seller. On the dependency that connects each instance to the collaboration, you can indicate the role played by the instance.



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:

- 6 ["Presentation of the State Machine Diagram", page 298](#)
- 6 ["Creating a State Machine Diagram", page 298](#)
- 6 ["States", page 300](#)
- 6 ["State Transitions", page 304](#)

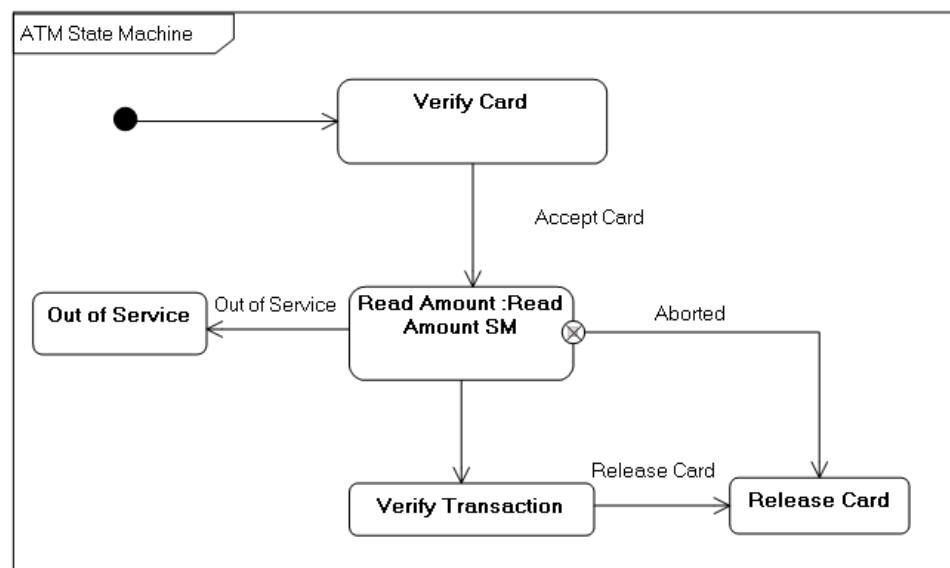
PRESENTATION OF THE STATE MACHINE DIAGRAM

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 V2** from the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit window, click the **State Machine** tile.
3. Right-click the name of the state machine concerned and select **New > State Machine Diagram..**
The diagram opens in the edit window.

The diagram is initialized by creation of a region. A region is part of a composite state or state machine which contains states and transitions and of which execution is autonomous.

STATES

A state is a condition or situation in the life of an object, during which it satisfies some condition, performs some activity, or waits for some event. A state represents an interval of time delimited by two events. It is a phase an object passes through during its life cycle.

Examples of object states

- A person can be:
 - Unmarried
 - Married
 - Divorced
- An item can be:
 - Available
 - In stock
 - At reorder level
 - Out of stock
 - etc.

Creating a State

To create a state in a state machine diagram:

1. Click the arrow associated with the **State** button of the object insert toolbar .
2. Select a state type.
3. Click in the diagram work area.
The **Add State** dialog box opens.
4. Indicate the **Name** of the state and click **Create**.
The state appears in the diagram.

State types

It is necessary to specify the state type at the time of its creation. It can be:

-  A normal state: has no sub-structure.
-  A composite state: comprises several states, described in the diagram.
-  A sub-machine state: calls the descriptor of a state machine described elsewhere. See ["Detailing Behavior of a State", page 302](#).
-  A final state

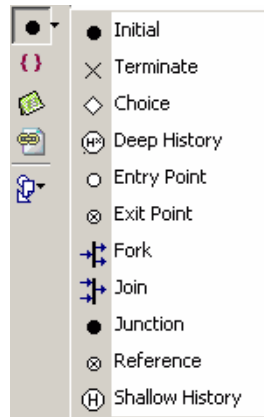
When you place a state in another state, it is automatically connected as a component of this state.

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", page 302](#).
- 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.

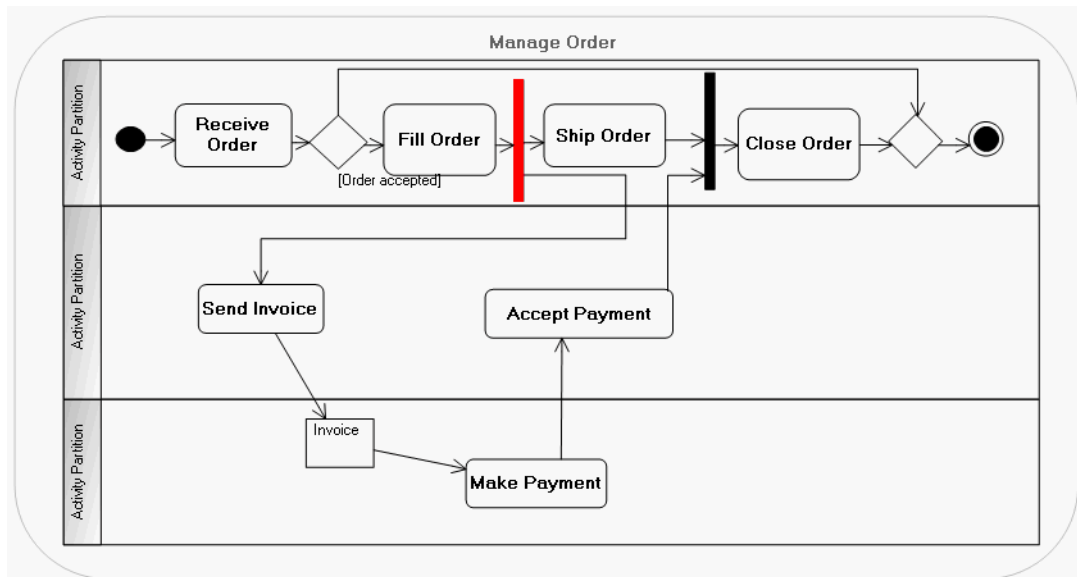
- 6 ["Activity Diagram", page 308](#)
- 6 ["Partitions", page 309](#)
- 6 ["Nodes", page 311](#)
- 6 ["Flows", page 313](#)

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 V2**, 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:

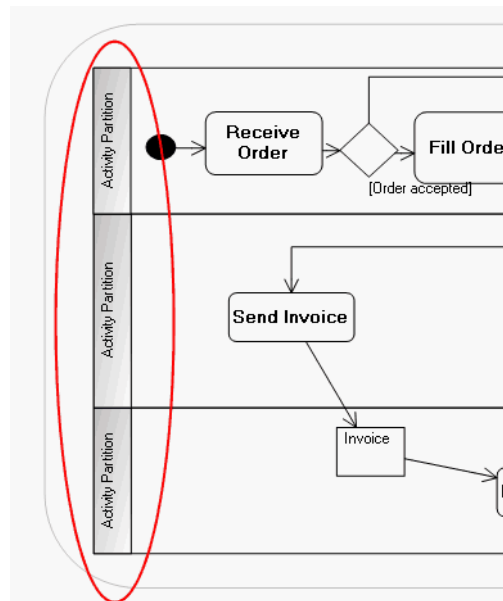
1. Right-click the package or the activity concerned.
2. In the pop-up menu that appears, click **New** > **Activity Diagram**.
The new activity diagram opens.

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", page 311](#)
- ["Parameter nodes", page 311](#)
- ["Control nodes", page 312](#)
- ["Object nodes: Input, Output and Exchange Pins", page 313](#)

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

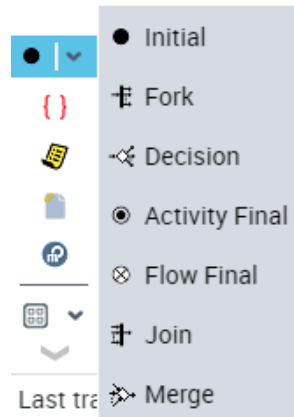
The parameter nodes of an activity describe the inputs and outputs of this activity.

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: :

- 6 ["Interactions", page 316](#)
- 6 ["Sequence Diagram", page 317](#)
- 6 ["Communication Diagram", page 329](#)
- 6 ["Interaction Overview Diagram", page 331](#)

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 V2** using the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit window, click the **Interactions** tile.
3. Click the **New** button.
4. Enter the name of the interaction and an owner if necessary.
5. 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**.

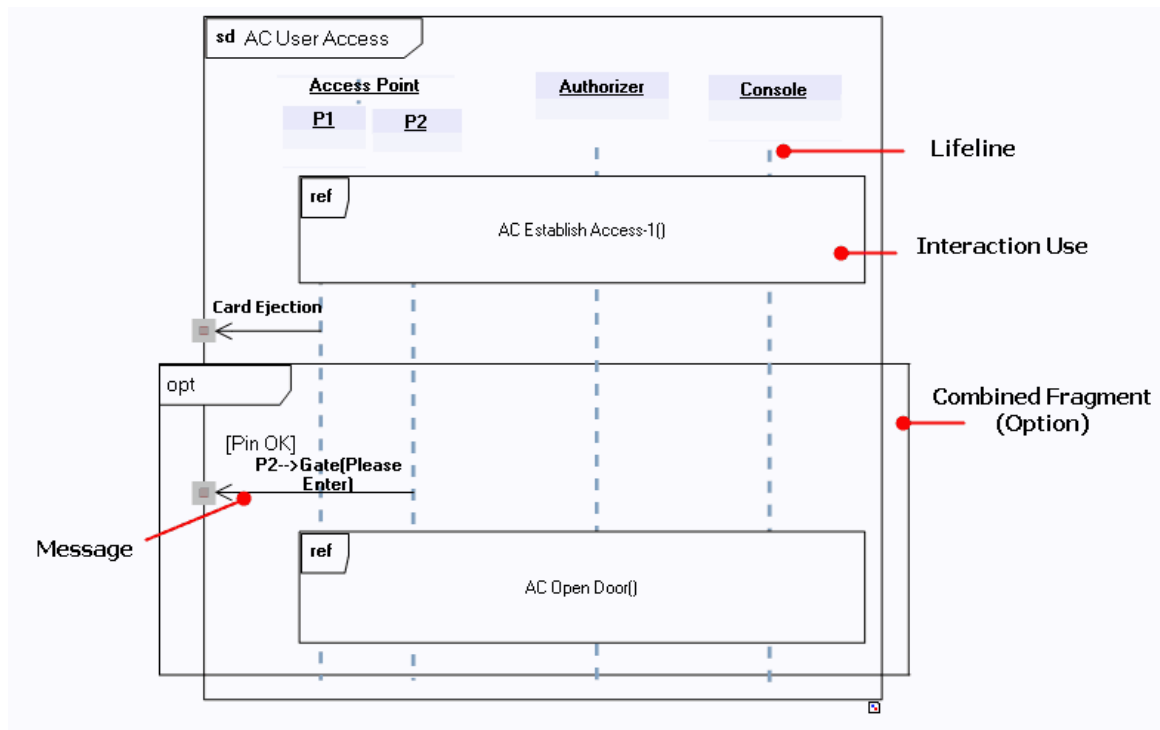
SEQUENCE DIAGRAM

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 V2** :

1. Right-click on an interaction.
2. In the pop-up menu that appears, click **New** > **Interaction Diagram**.

See also ["Creating an Interaction", page 316](#).

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.

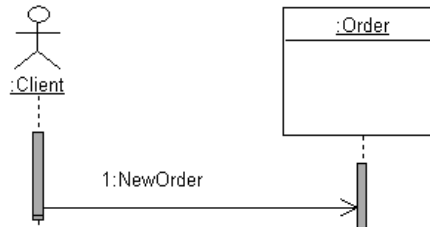
You can select the **Type** of the object (Actor, Class, etc.), specify the **Class**, **Actor**, etc. of which it is an instance, and indicate its *Stereotype*.

Messages

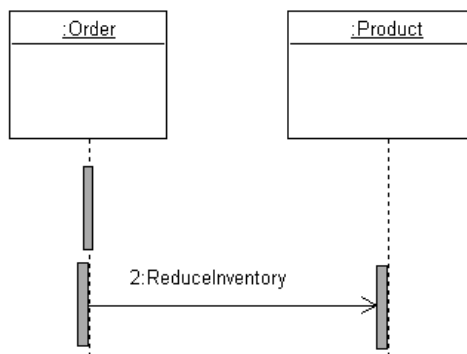
A message defines a particular communication between lifelines of an interaction. It specifies the sender and receiver via intermediate occurrence specifications, as well as the type of communication. This communication can be, for example, sending a signal, calling an operation or deleting an instance.

Examples of exchanged messages

- 1) The message sent by the "Client" actor to the "Order" class carries the "New Order" signal.



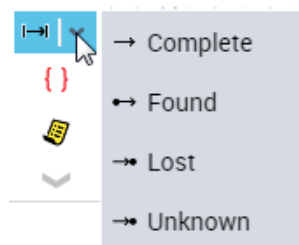
- 2) The message sent by the "Order" class to the "Product" class calls the "Reduce inventory" operation.



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.

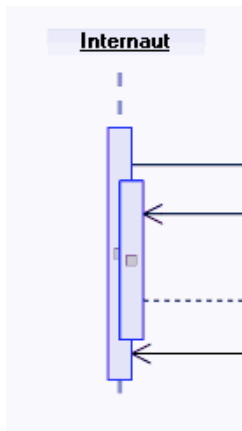
Message types

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

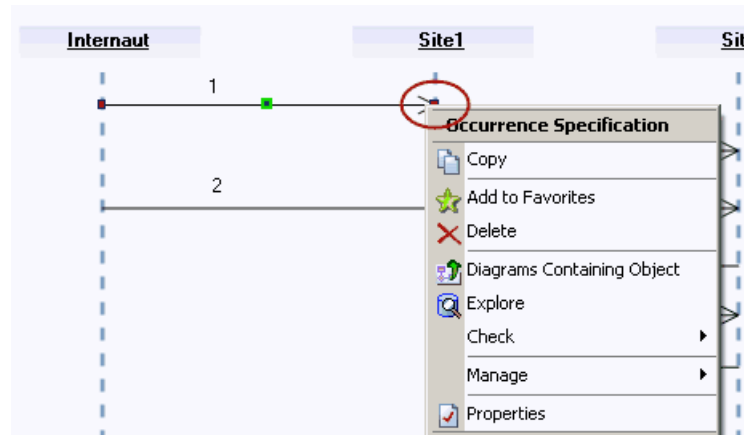
Creation of a message or an execution specification automatically creates occurrence specifications.

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.



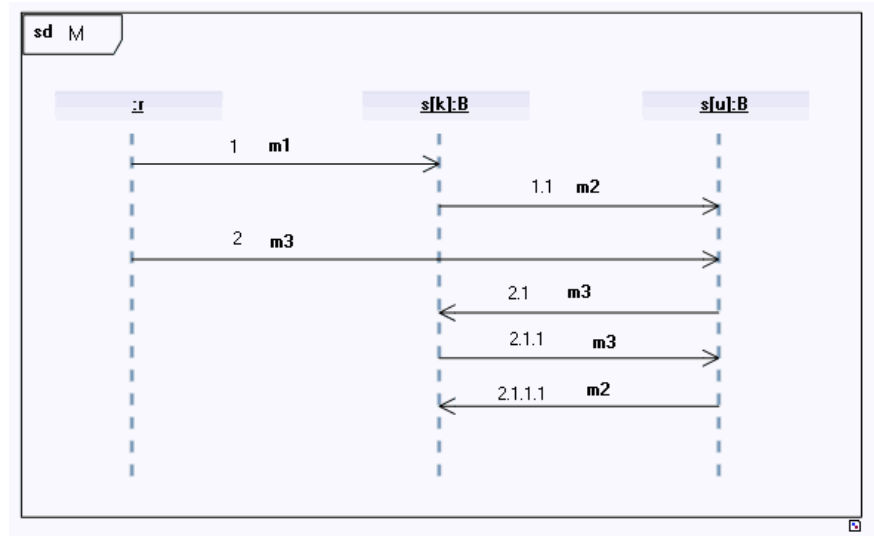
Calculating sequence numbers

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.

Example



You can manually modify the sequence number of a message in the message properties dialog box:

- > Select the **Characteristics** tab and change the value in the **Sequence Expression**.

When you restart calculation of sequence numbers, this updates sequencing according to the modifications made.

Combined Fragment

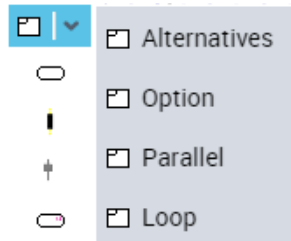
A combined fragment enables concise description of several execution sequences.

A combined fragment is defined by an interaction operator and the corresponding interaction operands.

Creating a combined fragment

To create a combined fragment:

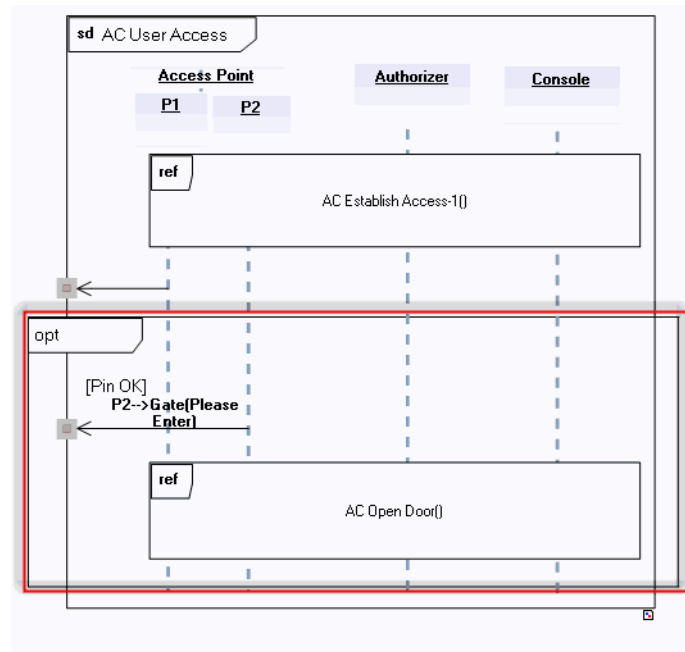
1. In the sequence diagram insert toolbar, click the **Combined Fragment** button.
You can associate different types of interaction operator to a combined fragment. The arrow at the right of the button offers shortcuts to four of these. See ["Interaction operator type", page 324](#).



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

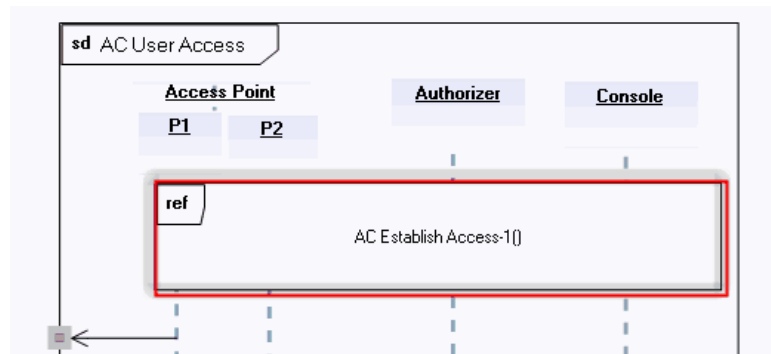
To create the interaction constraint that will condition the operand:

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.

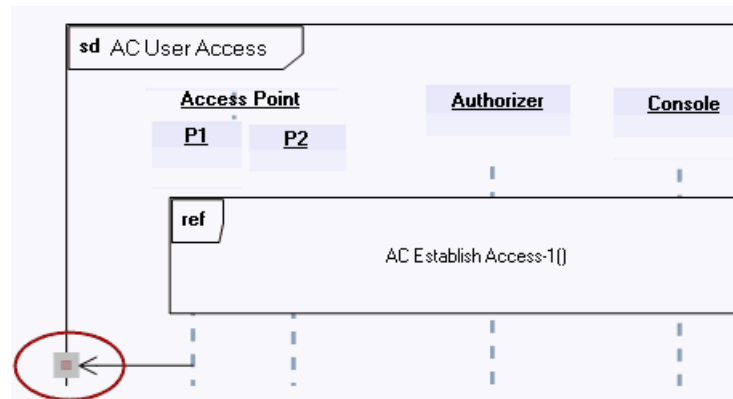
To align the argument with the interaction parameter called:

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.

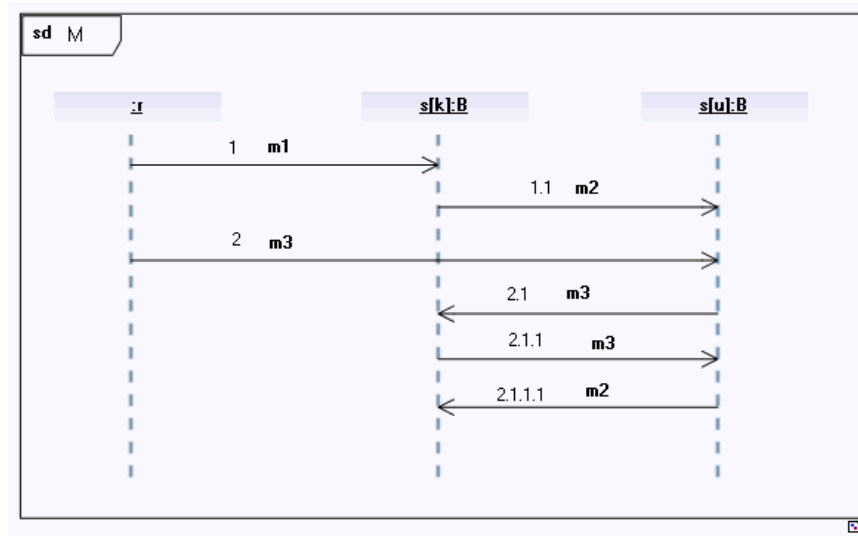
COMMUNICATION DIAGRAM

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

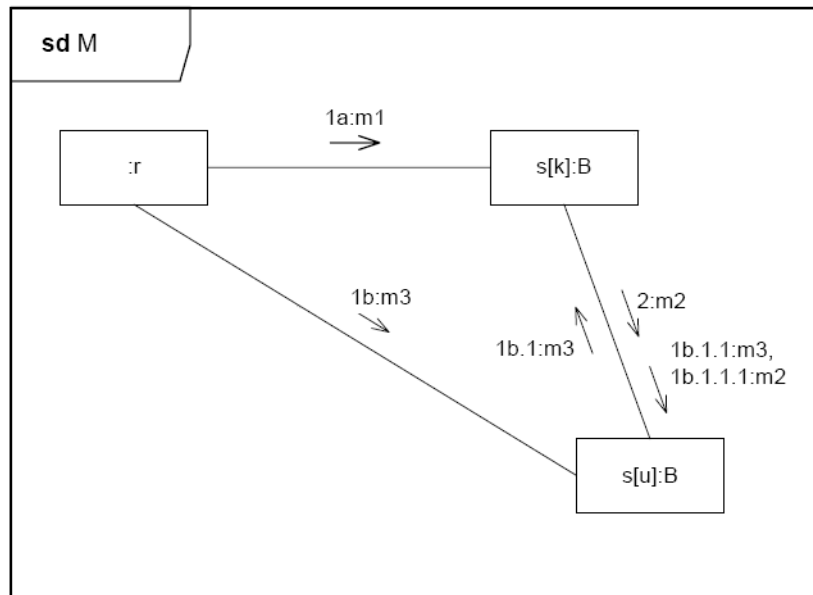


Diagram objects

Communication diagram objects are lifelines and messages transmitted by connectors.

When you connect two lifelines with a connector , the connector creation dialog box proposes messages that may be transmitted.

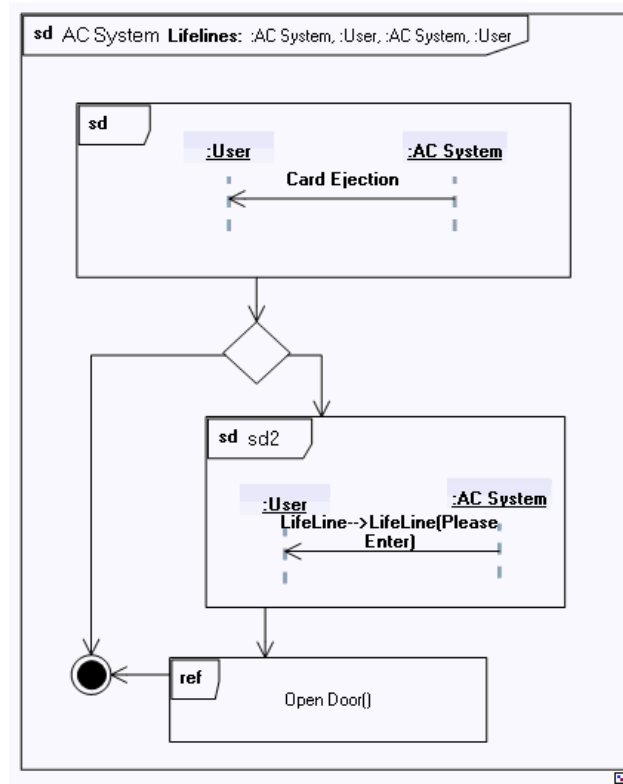
When the connector has been created, you can associate new messages in its properties dialog box, in the **Message** tab.

The sequence of messages is given by a sequence number associated with each message. See ["Calculating sequence numbers", page 321](#).

For more details on connectors, see ["Connectors", page 291](#).

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.

6 ["Presentation of the Deployment Diagram", page 332.](#)

PRESENTATION OF THE DEPLOYMENT DIAGRAM

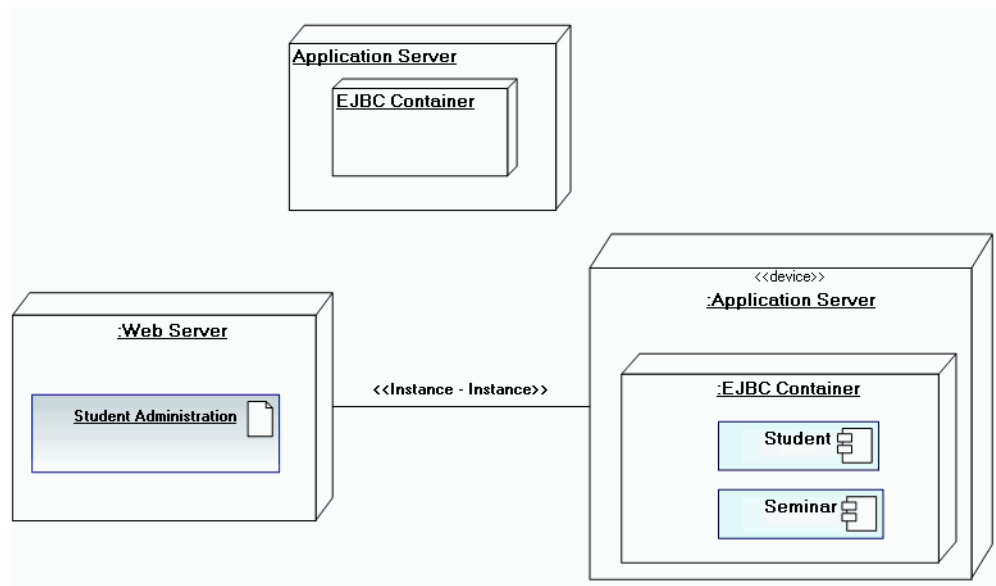
The deployment diagram complements the component diagram. It describes hardware resources (computer, router, etc.) in the system, and indicates distribution of components on these hardware resources.

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 V2**, a deployment diagram is created from a package.

To create deployment diagram with **HOPEX IT Architecture V2** from the **Software Design** navigation pane:

1. In the navigation pane, select **OO Implementation (UML)**.
2. In the edit area window, click the **Packages** tile.
3. Right-click the package in question.

4. In the pop-up menu that appears, select **New > Deployment Diagram**.
je ne le vois pas
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", page 295](#).

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.

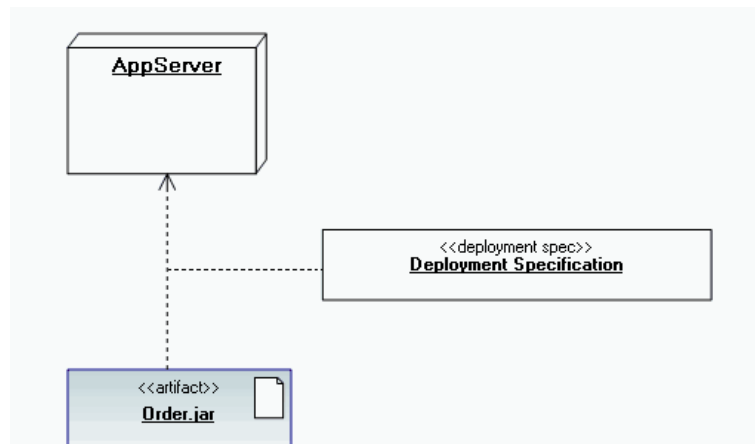
Deployment specification

Deployment specification enables indication of the characteristics that determine execution parameters of an artifact or component deployed on Node.

Configuration

The configuration button  enables creation of the link between a deployment specification and a deployment.

Example



APPENDIX: ATTRIBUTE TYPE



The following points are covered here:

- 6 ["Primitive Types", page 336](#)
- 6 ["Packages and Primitive Types", page 338](#)
- 6 ["Defining New Primitive Types", page 341](#)

PRIMITIVE TYPES

A primitive type is used to group characteristics shared by several attributes. Primitive types are implemented as classes.

To access primitive types in **HOPEX IT Architecture V2**, the administrator must import the "Information Architecture" solution pack. See ["Prerequisite: Importing the Primitive Types", page 336](#).

Prerequisite: Importing the Primitive Types

To access primitive types in **HOPEX IT Architecture V2**, the administrator must import the "Information Architecture" solution pack.

Preparing the import

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

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

Importing the solution pack

To import the solution pack:

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

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 January 1, 1970	
M-Bool	Boolean, equals 0 or 1	
M-Multimedia	Binary string	

PACKAGES AND PRIMITIVE TYPES

Packages

) *A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.*

The assignment of classes to packages imposes a rigid structure. As a class can belong to only one package, it is necessary to define client/supplier relationships so packages can use classes they do not own when they need to.

This is especially important for primitive type classes, because they will be used to define the attributes of other classes.

- *Rule: a class can belong to only one package.*

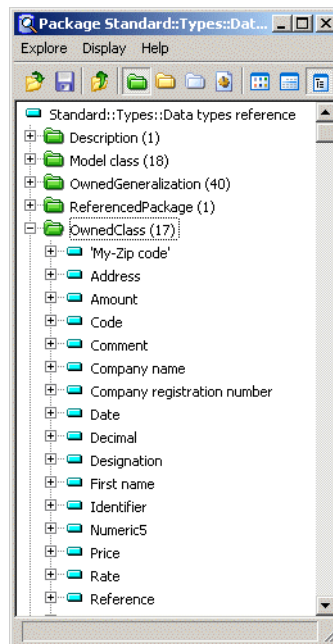
What primitive types are available for typing the class attributes depends on which package the class is in.

The type you can give to class attributes can only be primitive types defined for the package containing the class.

The accessible primitive types are public classes with the "Primitive Type" stereotype, that are contained in or are used by the package or the packages of which it is the client.

You can define a reference package (or several reference packages) containing the primitive types used by the enterprise. All the other packages are declared as clients of the reference package of primitive types.

In the example below, the "Data types reference" package contains the classes "Address", "Code", "Date", etc.



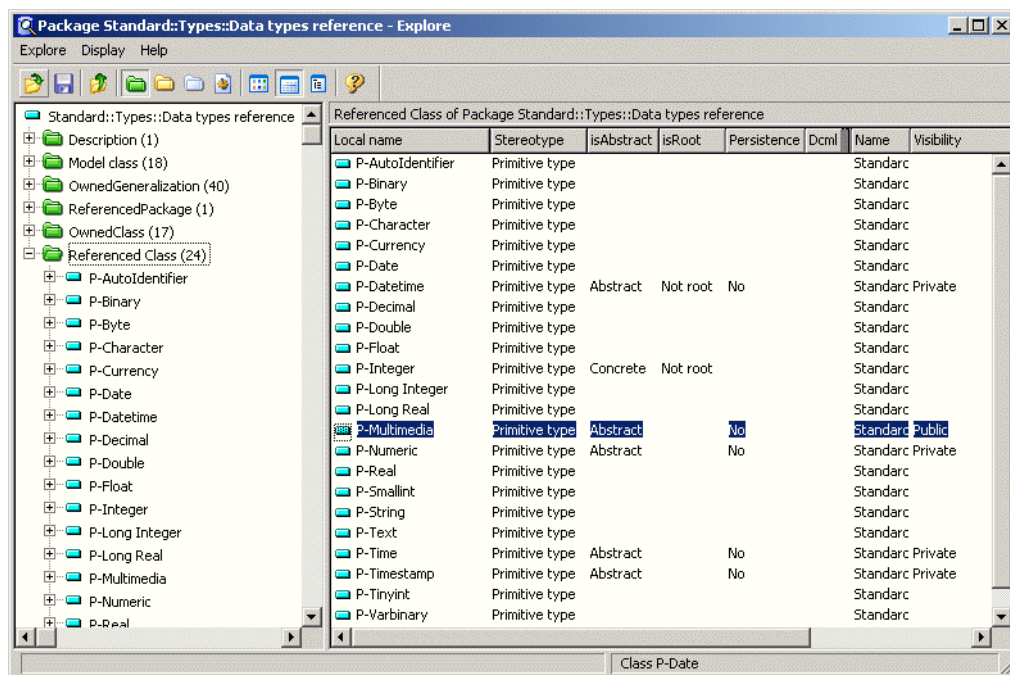
It is referenced by the packages "Library", "Order management", etc.

The class attributes for these packages can be typed using the types "Address", "Code", "Date", etc.

It is also possible to specify directly that a package uses a class contained in another package.

In the example below, the classes "P-Datetime", "P-Multimedia", "P-Numeric", etc. are used by the "Data Type Reference" package without being owned by that package.

Of these classes, only "M-Multimedia" is exported by the package for public use.



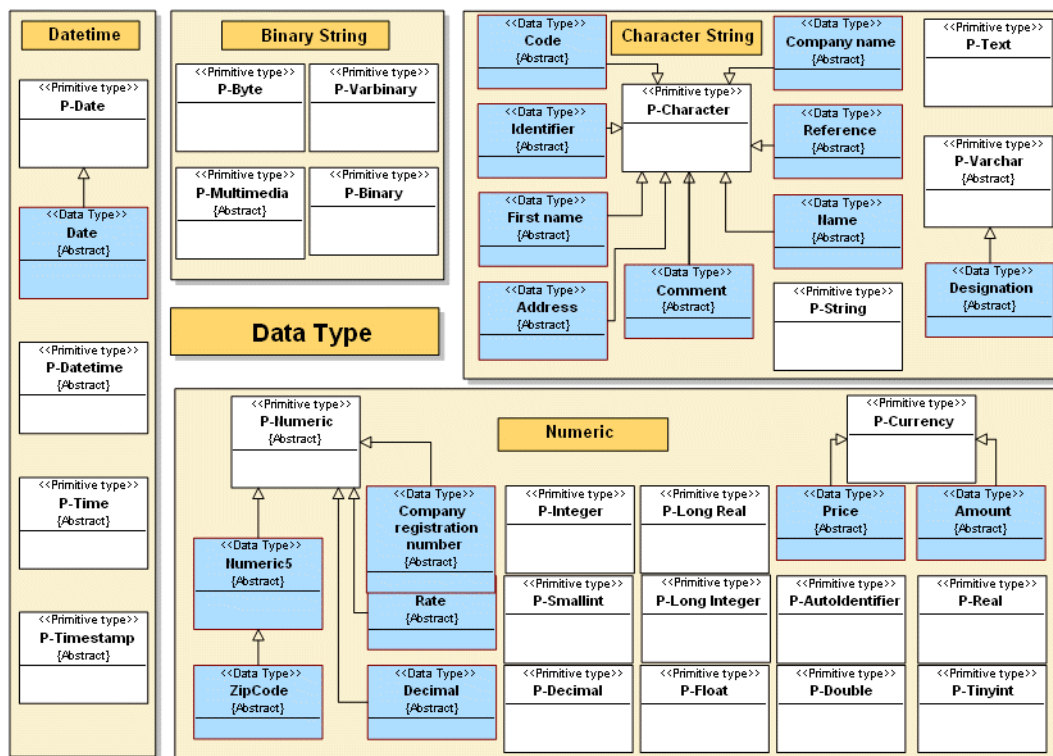
DEFINING NEW PRIMITIVE TYPES

New primitive types can be defined using a class diagram.

Depending on whether classes have been organized into packages, the class diagram can describe:

- A reference database.
- The package of reference types.

You can define your own primitive types by declaring them as subclasses of the standard primitive types, as shown in the example below:

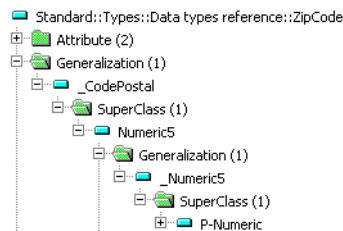


The primitive types defined as subclasses will automatically inherit the characteristics of their superclass. In particular, the datatype conversion rule for the superclass is applied to the subclass.

It is possible to specify a length and a number of decimal places for the subclass. These will be taken into account when generating the data types if they were not already defined for the superclass.

Inheritance can occur at several levels.

In the following example, the primitive type “ZipCode” is a specialization of the “Numeric5” type of length 5, which is itself a specialization of the standard type “P-Numeric”.

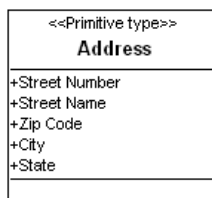


If the new primitive type is not defined directly or indirectly as a subclass of a standard primitive type, the conversion table that maps primitive types to column data types must be updated.

- A connection can also be directly defined between a type and the corresponding SQL datatype generated for each target DBMS without using the inheritance mechanism (see "Mapping between pivots and datatypes" in the **HOPEX Database Builder** guide).

Compound Primitive Type

You can define a compound primitive type by assigning to it a list of attributes.



Here the Address type is composed of number, street, zip code, city, and country.

The derivation of the Address attribute will produce these five columns.

It is possible to have several levels of compound types by assigning a compound type to an attribute of a compound type.

For example, the zip code can be broken down into the five main digits and the four-digit extension:

