

ArcStyler MDA-Business Transformer Modeling Style Guide For ARIS

For ArcStyler Version 3.x

Copyright

© Copyright September 27, 2002 Interactive Objects Software GmbH. All rights reserved.

The software described in this manual is provided by Interactive Objects Software under a license agreement. The software may be used only in accordance with the terms of the agreement.

This document or portions of it may only be transmitted or copied in units of unmodified complete pages or larger unmodified document segments. This copyright notice must be clearly visible at the beginning of each copy.

Interactive Objects Software GmbH registered trademarks: Interactive Objects, Convergent Architecture, ArcStyler, ArcExchange. All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Interactive Objects Software GmbH.

Basler Strasse 65

79100 Freiburg, Germany

<http://www.io-software.com>

Tel:+49 761 40073 0

Fax: +49 761 40073 73



Table of Contents

- Chapter 1. Introduction - From Process Models to Executable Applications 9**
 - ArcStyler MDA-Business Transformer for ARIS 9
 - ArcStyler, ARIS and Model Driven Architecture 9
 - Benefits for ARIS Users 11
 - Note to ArcStyler Users 12

- Chapter 2. Objectives of the Modeling Guide 13**
 - Modeling Objectives 14
 - Target Group 14
 - Benefits 14
 - Process Model 15

- Chapter 3. Modeling Style for eEPCs in ARIS Toolset 17**
 - Model Elements Used 17
 - Attributes Used 18
 - Group Structure in the ARIS Explorer 18
 - Updating Models not Created According to the Modeling Style 19
 - Model Element Naming Conventions 19
 - Function 20
 - Role Concepts and Specialization/Generalization 20
 - Role Concept 20
 - Specialization 20
 - Event 21
 - Object State (Product/Service) 23
 - Operators 23
 - Attributes 23
 - Class 24
 - Cluster 24
 - Entity Type 24
 - Organizational Unit 24

Organizational Unit Type	24
Position	24
Connections	24
Storage of the eEPCs and the Model Elements	25
Component	25
eEPC	25
Assignment of Models	25
General Information on Model Elements	25
Function	26
Special Case: Comparison Function	26
Resources	26
Class	26
Cluster	26
Entity Type	26
Attributes	26
Event	26
Object State (Product/Service)	27
Operators	27
Organization	27
Organizational Unit	27
Organization Unit Type	27
Position	27
Connections	27
Unambiguous Process Logic	28
Using Events	28
Processes with Several Start Events	30
Process Branches and Joins	31
Modeling the Correct Correspondence of Branches and Joins	32
Problematic Cycles	33
Concurrency	34
Verifiers	35
Model Element Mapping (eEPC → UML)	36
Model Element and Attribute Mapping	37
Multilingual Models	39



Before You Begin

This guide serves as a modeling guide for the ArcStyler MDA-Business Transformer for ARIS.

What Is in this Guide

The following chapters appear in this document:

- Chapter 1, “Introduction - From Process Models to Executable Applications”
- Chapter 2, “Objectives of the Modeling Guide”
- Chapter 3, “Modeling Style for eEPCs in ARIS Toolset”

Who Should Read this Guide

This guide is designed for ARIS users (managers, process owners and organization professionals) who analyze, design and optimize business processes in the form of models which are to be automatically transformed into executable applications. With the modeling style for the ArcStyler MDA-Business Transformer for ARIS, eEPCs can be modeled in such a way that they can be automatically transformed into UML. IT specialists can use these models for the automated development of software systems.

Related ArcStyler Documents

Other directly related ArcStyler documents are:

- *ArcStyler Modeling Style and User’s Guide*
Comprises a style guide for the technical modeling of components and their generative projection to a particular runnable system infrastructure. The focus lies on modeling and generating Enterprise JavaBeans. Moreover, the guide shows how to work with the generated sources and describes the build support provided by the ArcStyler.

- *ArcStyler MDA-Cartridge Guide For BEA Weblogic Integration Server 2.1*
Describes the features of the ArcStyler technology projection cartridge for the BEA WebLogic Integration Server 2.1. It explains the ArcStyler support for model-driven development of Enterprise JavaBeans systems for the BEA WebLogic Server.
- *ArcStyler Accessor Guide*
Covers the general design and usage features of the patent-pending ArcStyler Accessor Framework. This framework supports the model-based, object-oriented development of external interfaces for software systems including multi-channel B2X Internet interfaces. Moreover, it provides a detailed description of the JSP/Servlet technology projection provided by the ArcStyler Accessor Cartridge, which supports automatic generation and deployment of JSP/Servlet-based Web applications from UML models.
- *ArcStyler MDA-Business Transformer Tutorial*
Describes the sample transformation of an ARIS *extended Event-Driven Process Chain* (eEPC). The model follows the *ArcStyler MDA-Business Transformer Modeling Guide for ARIS* and uses methods according to ARIS Best Practices to model assigned processes.



Introduction - From Process Models to Executable Applications

ArcStyler MDA-Business Transformer for ARIS

The integration module for ArcStyler and ARIS serves as a bridge between ARIS, a process engineering tool designed to graphically create business process models, and ArcStyler, a comprehensive architectural IDE (Integrated Development Environment) for application engineering according to the Model Driven Architecture (MDA) as standardized by the Object Management Group (OMG).

By integrating these two tools, the ArcStyler MDA-Business Transformer for ARIS also provides a closer link between two distinct units or departments within an organization: Management and the organization department on the one hand and the IT department on the other hand, i.e. this technical integration in itself is a process optimization. Managers, process owners and organization professionals can analyze, design and optimize business processes in the form of models. IT specialists can use these very models for the automated design and development of software systems - a seamless and tool-supported process. Business requirements are transformed into software systems without information loss by means of a full-coverage chain of tools within the framework of a consistent architectural style.

ArcStyler, ARIS and Model Driven Architecture

Model-based process engineering and Model Driven Architecture have a number of common objectives. Both aim at the abstract, structured description of processes in a modeling language to create easy-to-understand and well-documented models. In both cases, the initial models are independent of implementation details. For example, a process model abstracts from the individual persons who will have to implement the process in an organization. A high-level MDA application model abstracts from the implementation technology that will eventually be used to create the final program, the executable application. In the MDA paradigm, such a model is referred to as a

platform-independent model (PIM). This PIM describes the business logic of the application to be built.

The MDA development process with ArcStyler is highly automated. In this process, implementation-specific detail is added to the PIM by means of so-called model-to-model transformations and a final model-to-code transformation. These transformations are performed by means of a template-based mapping process controlled by ArcStyler's so-called MDA-Cartridges. Depending on the implementation platform desired (e.g. J2EE or .NET), different MDA-Cartridges are used that can be simply plugged into the ArcStyler. This way, it is possible to automatically generate technology projections of one and the same platform-independent model to several implementation technologies. This procedure results a number of important benefits:

- Developers can concentrate on the design of the business logic of the application to be built without having to invest great effort in the specifics of the implementation technology to be used as these are automatically added by ArcStyler.
- The platform-independent model is completely decoupled from any implementation technologies, i.e. its lifecycle is independent from the lifecycles of the implementation technologies making it is future-safe.
- The high degree of automation involved in the development process reduces costs.
- The models are a lot easier to read and understand than code, which makes communication easier for everybody involved in the development and reduces the documentation effort required; in fact, the model is the documentation.

An ARIS model that meets certain requirements is an MDA-compliant PIM, but there is also an essential difference between the business process-oriented ARIS models and the implementation-oriented ArcStyler MDA models: the language. ArcStyler and MDA use the standardized Unified Modeling Language (UML) to enrich an activity model with implementation-specific technical properties and turn into an application. While an ARIS model may be a lot easier to understand for managers and process engineers, UML is an implementation-oriented language that is required for MDA-compliant application development. This means that an ARIS model needs to be converted into UML before it can be further processed with ArcStyler - which is exactly what the ArcStyler MDA-Business Transformer for ARIS does automatically.

Benefits for ARIS Users

Many ARIS process models are designed to be implemented as application programs. They serve as the documentation for the IT department and clearly describe the individual requirements to be turned into program logic. However there was little or no tool support in this transformation process. And in quite a few cases, the corporate IT was unable to implement the model exactly as it was. This was due to a variety of reasons such as organizational or technical problems. In such cases, the technical implementation often differed from the ARIS model, and communication problems between the IT and the organization departments or other factors prevented the ARIS models from being updated accordingly, i.e. the documentation and the application were no longer in sync.

A typical example of a technical aspect that may lead to problems in the implementation of an ARIS model is the issue of concurrency. Concurrency means that after a process branch there are several instances (one per branch) of a process, as opposed to a single instance prior to the branch. In the runtime environment, these concurrent processes must be technically synchronized so that the complete process can be properly terminated and no “dead processes” exist in the system. While this may not cause any problems at the ARIS model level, it may lead the application to crash if it is not handled properly.

With the ArcStyler MDA-Business Transformer for ARIS, you can model your eEPCs in such a way that it can be automatically transformed into UML and that all the problems you and your IT people may have encountered before are solved at the ARIS model level. This means that the ARIS model is no longer “only” a documentation of the processes to be implemented, but it serves as its very basis, as the PIM.

To do so, your model has to follow certain conventions. These conventions are defined in the so-called Modeling Style. A Modeling Style defines the vocabulary, the semantics of the terms and the relationships between them as well as the consistency criteria for the corresponding models. It anticipates ambiguities, inconsistencies and contradictions during the modeling process and defines appropriate solutions. This way, any ARIS model that complies with the Modeling Style can be directly transformed into a UML model for further processing with the ArcStyler.

The modeling style supports eEPCs and the appropriate model elements (such as Component Groups, Functions, Events, Operators, Object States, Classes, Attributes, Entity Types, Organizational Units, Organizational Unit Types, Positions, etc.). ArcStyler automatically maps these elements to the corresponding UML elements. While there are a few restrictions to observe, the discipline involved in adhering to the Modeling Style will pay off by orders of magnitude. Your ARIS business process model is a PIM that can be directly processed by ArcStyler so that it is

possible to automatically convert the business logic into an implementation-oriented modeling language, bridging the gap between business dimension and implementation. This enables the automatic generation of optimized code for the target implementation platform for applications that are ready to run right away.

Note to ArcStyler Users

As an ArcStyler user, you will most likely be familiar with business component modeling from working with ArcStyler's Business Object Modeler C-BOM. C-BOM uses CRC cards to model the business dimension of the application. The ArcStyler MDA-Business Transformer for ARIS allows you to perform the similar tasks with ARIS.



Objectives of the Modeling Guide

A modeling style allows the user to create models at a specific levels of abstraction and with a specific granularity. It defines the vocabulary, the semantics of the terms and the relationships between them as well as the consistency criteria for the corresponding models. The extensions to the basic methodology introduced by a modeling style improve the usability of a model within the framework of an MDA-centric software development process. Using a well-defined vocabulary and well-defined, refined semantics improves the legibility of models and their re-usability in diverse contexts. In particular, a precise modeling style is provided to facilitate the tool-supported, automatic transformation of models into models with different levels of abstraction and into executable code. All information available in a given stage of the software development process can be completely represented by means of and in compliance with the modeling style.

The modeling style described in this document is intended to support the designer of a process model, i.e. the business modeler, in implementing models in compliance with Model Driven Architecture (MDA, an approach standardized by the Object Management Group) in the context of a comprehensive and holistic technology and tool chain.

The modeling style supports the development of an object-oriented process model.

It anticipates ambiguities, inconsistencies and contradictions during the modeling process and defines the appropriate solutions. The modelers should carefully study the modeling style prior to starting the modeling process. During modeling, they can use the modeling style as a reference work. Therefore, the most suitable medium is a text document with illustrations which is available in both online and offline modes.

The modeling style assumes knowledge of object-orientation and of the eEPC (extended event-controlled process chain) as well as the tools used. The modeling style is not a comprehensive description of the methods and the functionality of the tools used.

The modeling style is designed for:

- The graphical modeling tool ARIS 6.0, referred to as ARIS below.
- The Architectural IDE ArcStyler 3.0 with all the tools it comprises, referred to as ArcStyler below.

If the tools are updated, this modeling style may have to be adapted accordingly in terms of coverage and degree of detail.

The modeling style may be adapted to specific requirements in given projects to the degree that such modifications are supported by the tools used. In such cases, a project-specific version management scheme should be set up.

Modeling Objectives

The objective of modeling is the creation of models allowing for the implementation and documentation by means of automated tool support through ArcStyler in compliance with the MDA standards. This implementation is performed by means of refinements of the models and the ultimate generation of a standards-conform runtime infrastructure. The modeling is supported by the modeling style described in this document. The models created with ARIS according to this modeling style are platform-independent models (PIM) in terms of OMG MDA with reference to concrete implementation technologies (such as WfMS, J2EE, .NET). The modeling style does not presuppose any specific implementation technology.

Target Group

ARIS users who want to transform their models into an executable system for a specific runtime environment, using the MDA-compliant tool support and automation functionality provided by ArcStyler.

Benefits

- Standards-based, future-safe process with an Architectural IDE for MDA.
- Superior model quality due to well-defined, extended semantics, early detection of inconsistencies, high degree of re-usability.
- Tool-supported and automatic transformation of the model into a model with a different level of abstraction or a different view and into implemented infrastructure.

Process Model

After the business domain has been defined, delimited and structured, ARIS extended Event-Driven Process Chains (eEPCs) can be modeled within this domain on the basis of the modeling style. ArcStyler converts these models into UML models by means of model-to-model transformations. Implementation attributes are added to these UML models, and then ArcStyler generates runtime infrastructure (such as J2EE/EJB, .NET, Host, EAI, SAP, B2C, Web Services, etc.).





Modeling Style for eEPCs in ARIS Toolset

Within the context covered in this document, a process describes a network of actions with logical relationships defined by predecessor and successor relationships. A process can consume resources, generate resources or use the services of another resource. A process always consumes time.

Model Elements Used



Just a few of the numerous model elements and model element types provided by ARIS are sufficient for modeling according to this modeling style. The eEPC is the only model type implemented by ArcStyler according to the process described in this document. Future releases may support other model types. Process chain diagrams (PCD) must be converted to eEPCs in ARIS if they are to be used for UML transformation and further processing in ArcStyler. In terms of the modeling style, an eEPC uses the following ARIS model elements:

- Function
- Event
- Operators
 - AND
 - OR
 - XOR
- Object State (Product/Service)
- Resources
 - Class
 - Cluster
 - Entity type
- Attributes
- Organizational unit
- Organizational unit type
- Position
- Connection (all connections allowed between the model elements used in the eEPC)

By default, ArcStyler only processes the above model elements.

For easier modeling, a special methods filter may be defined which contains only the above elements. The entire method of the ARIS standard comprises the required elements and may therefore also be used. The best approach is to make the decision for a project-specific method filter prior to starting the modeling work.

Attributes Used

By default, ArcStyler processes the model element attributes listed below. These are attributes common to all model elements:

- Name (not for connection)
- Description/definition

An additional free attribute with the designation *Join Semantic* can be defined in the case of operators.

By default, ArcStyler does not process other attributes.

It is possible to define a specific method filter for easier modeling so that only the attributes listed above are accessible.

Group Structure in the ARIS Explorer

The ARIS model database can be structured by means of logical groups. The group structure is created below the main group in the model database.

Two groups should always be created below the Main Group:

- Process hierarchy
- Component catalog

These groups allow you to structure your model. They may be subdivided into additional sub-groups.

The group *Process hierarchy* allows you to hierarchically structure your business processes. The group names should therefore reflect the appropriate business tasks. The group structure and the

component catalog may be used to define access rights in development processes involving several modelers working on the model. This way, it is possible to specify persons responsible for the process and the components. In such a case, the sub-groups should correspond to work packages for the distributed teams of modelers in order to facilitate the administration of rights.

The identification of process levels has several aspects. It is difficult to define rules for the creation of process levels and level characteristics which are not only governed by formal aspects, but also in terms of content. A proven approach consists of handling these issues in a project-specific way.

Further parallel or parent groups (for the process hierarchy and the component catalog) may be created, e.g. for navigation models or different ARIS views. However, these groups and their model contents and model element contents are not processed by ArcStyler.

Updating Models not Created According to the Modeling Style

This section describes how to update models which were not created in compliance with the modeling style so that they can be processed with ArcStyler.

Create a group structure according to the above rules in addition to the existing group structure. Create variants of the existing models in the appropriate new groups. These models must be modified according to the rules outlined in the following sections. This way, it is always possible to compare the original model and the processed model. ArcStyler only processes the variants.

Model Element Naming Conventions

The modeling style does not stipulate any uppercase/lowercase notation or underscores in compounds. While (project-specific) guidelines improve the model quality, the notation does not affect the further processing of the models all the way to executable code.

If no other information is given, the syntax rules relate to the names of the corresponding model elements. The attribute **Description/Definition** may be used for additional explanations as required.

Function

In an object-oriented approach, operations presuppose the existence of attributes or relations upon which the operations act or that methods are made available by the object itself. The process-oriented eEPC is to be transformed into a process- and object-oriented representation. Therefore, in the notation of a function, you have to make sure that it refers to an object (instance of a component) or to the attribute of such an object.

Notation for functions relating to components:

- **Verb.ComponentName** (infinitive)
 - Example: **Create.Order**

Notation for functions relating to attributes:

- **Verb.ComponentName.AttributeName** (infinitive)
 - Example: **Create.Order.Number**

A number of operations are used in almost every class (lifecycle operations such as **create**, **find/call**, **assign**, **edit**, **save**, **copy**, **delete**, **cancel**, **undo**, etc.).

The lifecycle operations are included in models designed for the purpose of software development/generation, in particular the following operations:

- **Create** (for instantiation of objects)
- **Assign** (for creating relations)

Role Concepts and Specialization/Generalization

The following sections will first provide definitions of these terms and then propose a simplified modeling approach without a differentiation of the two concepts. The explanations are provided to solve problems modelers may have who are familiar with these concepts.

Role Concept

A role defines the way a component **A** is perceived by a component **B** related to component **A**.

In terms of process modeling this means that a role process is used by a base process in a specific context. The role process is limited to the aspects relevant within this context.

Specialization

Within the framework of this modeling style, the specialization consists of a differentiated extension of an eEPC.

Since it is difficult to distinguish between role and specialization during the phase of eEPC modeling, the two cases are treated in the same way syntactically:

- **Verb.Role/Specialization**
 - Example: **Assign.Material** specializes **Assign.Resource**; **Assign.PayeeBankAccount** is a role of **Assign.BankAccount**.

A component must be created in the appropriate group for the role or specialization. Differentiation and refinement is performed in UML after the transformation by ArcStyler.

Event

Background information: Modeling events in eEPCs

In an IT runtime environment, it is difficult to interpret an eEPC event as an event. It can be interpreted as a precondition/postcondition of a function and as a switching condition at a transition. In this sense, an eEPC event corresponds to an object state condition in an IT runtime environment.

Events are used to represent the process logic in eEPCs. By preceding a function with events in the process run, it is possible to specify when it occurs logically.

An event can be described with different degrees of detail and at various levels:

- Value description at the level of a single or several attributes of a component.
 - Example (“_” represents a blank):

House.Height = _100

(House.Height <= _10_AND_House.Width = _200)

- Description of the existence of a relationship (reference) between instances.
 - Example:

Customer.assigned

- Extension of one or several components.

- Example:

Number(Customer)_=_30

(Number(Customer)_>=_10)_AND_(Number(Product)_=_20)

Modeling of elementary events.

Events exclusively define elementary states (description of an attribute value, a relation between components or description of a property of the extension of a component; also see the example above). A negating complementary event can be described for each event:

House.Height =_[100]

Complement:

NOT

House.Height =_[100]

The following sections summarize the general notation of events:

- Relating to an ATTRIBUTE:
 - **ComponentName.AttributeName.Participle** for Boolean attribute
 - **ComponentName.AttributeName_ComparisonOperator_[ComparisonValue]** for attribute with ordered value range
 - **ComponentName.AttributeName_ComparisonOperator_ComponentName.AttributeName**
- Relating to a component:
 - **ComponentName.Participle** for component state
- Relating to extension of a CLASS
 - **Number(ComponentName)_ComparisonOperator_[ComparisonValue]**

To model complex events, link elementary events by means of logical operators.

As opposed to the notation of elementary events, the notation of complex events can be relatively free.

Defined start and end events must be created for each eEPC. This way, an eEPC indicates which operations (in this case functions) need to be performed when (logically) for the process transition from the start state to the end state.

The events complement each other, i.e. as a rule, after the execution of the operation, either the end state must have been reached, or the start state must not have changed.

The names of the two events correspond to the name of the eEPC; the verb is used in the participle form.

A **NOT** is added to the name of the start state.

Example:

- Signifier eEPC:Customer.assign
Signifier start state:

NOT

Customer.assigned

- Signifier end state:
Customer.assigned

Object State (Product/Service)

The notation of the object state is identical to the notation of events with reference to components:

ComponentName.Participle

Operators

- AND
- OR
- XOR

The free attribute *Join Semantic* of the **OR** operator provides the values **First Come** and **Wait for all active**.

Attributes

Notation: **ComponentName.AttributeName**

ComponentName and **AttributeName** are **Nouns**.

Class

Notation: **Noun**. The class name corresponds to the appropriate component name.

Cluster

Notation: **Noun**. The cluster name corresponds to the appropriate component name.

Entity Type

Notation: **Noun**. The entity type name corresponds to the appropriate component name.

Organizational Unit

Notation: **Noun**

Organizational Unit Type

Notation: **Noun**

Position

Notation: **Noun**

The position is used as a user role, i.e. it responds to the question which user (as a role) accesses which function.

Connections

Loop iterators are described in the connection role of the connection and positioned.

Possible iterators:

Attribute of an object

Example: **PaymentOrder.NumberSinglePaymentOrder**

Counter of a class extension

Example: **Number(Customer)**

Storage of the eEPCs and the Model Elements

Component

All components used in the model are stored in the component catalog.

One group is created for each component. The group has the same name as the component. The ARIS object definitions and models belonging to the component are stored in this group. Within the model, the component is unique. The component name is the unique signifier.

eEPC

The business context for a group is often defined and limited by the eEPC of the same name.

An eEPC is used to represent the internal view of an operation. The operation determines the context and, by implication, the scope of the representation.

eEPCs are created in the group of the corresponding component in the component catalog. In addition, a shortcut is generated in the corresponding process hierarchy group.

There may be situations in which different functions call the same process (for example, in the case of roles or specializations which are defined in this modeling style, but which are not differentiated during modeling).

Assignment of Models

An eEPC is always created as an assignment for a function (except at the top level). This way, the model is automatically stored in the same group in the ARIS Explorer that contains the function (in the group of the component to which the function belongs).

The model name automatically corresponds to the function name (ARIS functionality).

General Information on Model Elements

A model element is always created in the Explorer in order to assure that this ARIS object is stored in the proper group (group which is named after the component).

A model element should not be created in the diagram by means of the modeling toolbar. If you do so, the system cannot assure that the ARIS object is stored in the proper folder of the ARIS Explorer.

Function

A function is stored in the group of the component whose name is a part of its name.

Special Case: Comparison Function

Since a comparison always involves two things, it is difficult or impossible to specify to which component the comparison is to be assigned. In addition, the comparison as such is not interesting in terms of the operation, but only the result of the comparison which determines the further process.

In the process, a comparison is used to prepare a branch. There must be a condition at each branch operator (in particular in the case of **OR** and **XOR**).

Comparisons are therefore not modeled as functions but as links between events. See “Event”.

Resources

Class

A class cannot be created unless a group exists. Therefore, when you create a class, you must also create a group in the Explorer.

Cluster

A cluster cannot be created unless a group exists. Therefore, when you create a cluster, you must also create a group in the Explorer.

Entity Type

An entity type cannot be created unless a group exists. Therefore, when you create an entity type, you must also create a group in the Explorer.

Attributes

Attributes are created and stored in the group of the component to which they are assigned. The data types are defined after the transformation.

Event

An elementary event is stored in the group of the component whose name is a part of its name.

Results of comparisons and complex events are created and stored in the group *System States* (sub-group of the of the component catalog) unless their name uniquely indicates the component to which they belong.

Object State (Product/Service)

An object state is created and stored in the group of the component whose name is a part of its name.

Operators

Operators have to be created in the models themselves. They are automatically stored in the group of the process in which they are used.

Organization

Organizational Unit

An organizational unit is created and stored in the group of the component whose name is a part of its name.

Organization Unit Type

An organizational unit type is created and stored in the group of the component whose name is a part of its name.

Position

A position is created and stored in the group of the component whose name is a part of its name.

Connections

Connections have to be created in the models themselves. They are automatically stored in the group of the process in which they are used.

Unambiguous Process Logic

Modeling processes in general and in ARIS Toolset in particular offers a certain degree of freedom. For an eEPC to be able to serve as the basis for automatic or manual implementation, the degree of freedom must be discussed with the appropriate experts and limited. This reduces the risk of misinterpretation of the requirements on the part of the developers involved in the implementation. Since the decisions concerning the reduced degree of freedom are caused by process modeling considerations, it is strongly recommended to provide a model in the tool (in this case ARIS Toolset) of the appropriate experts. This model provides clear instructions and guidelines for everybody involved in the process as well as a documentation for everybody having to understand and/or implement the process.

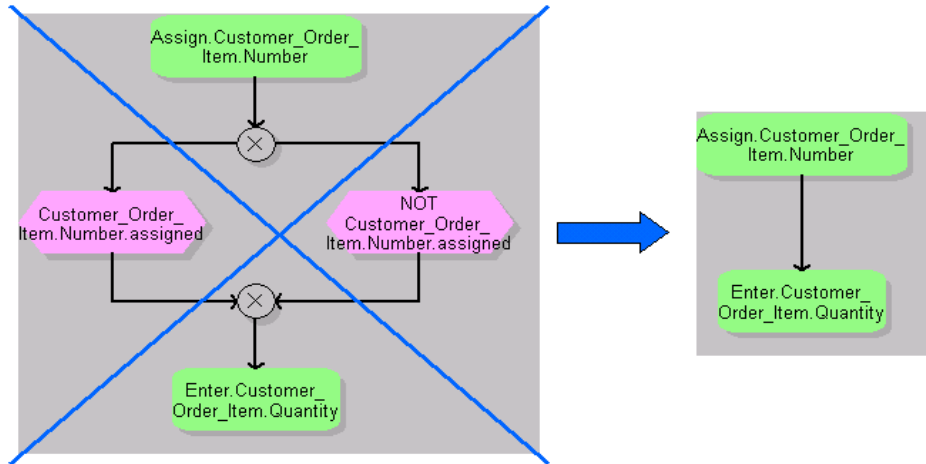
The following sections describe the procedure for creating a clear and implementable process logic.

Using Events

A function begins with a start event and ends with an end event.

A number of events do not have any influence on the control flow of the process. Therefore, there is no point in modeling such events.

In order to avoid unnecessary complexity in the graphical representation of an eEPC, only those function-related events are represented which are a logical prerequisite for the continuation of the process in terms of reaching the end event of the process.



In this example, the alternative results of a function do not influence the process logic; therefore, they are not represented.

Figure 1: Trivial Events.

A loop is represented by a looping process path connected by the main process path by means of an XOR connector. The iterator determining the number of loop runs is added to the path.

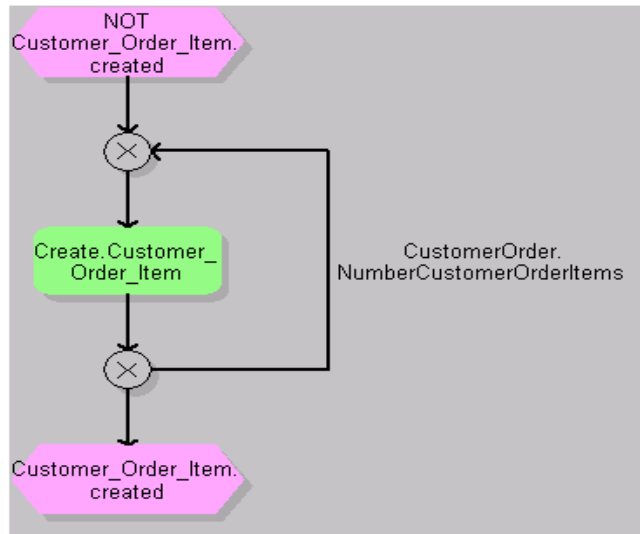


Figure 2: Loops.

Processes with Several Start Events

If the process illustrated is to be implemented, the following question needs to be answered: What happens, if several start events occur simultaneously? Solution: The events are treated in the same way as if they had been preceded by an XOR operator. Each start event triggers a new process instance, i.e. start events occur as alternatives for each process instance.

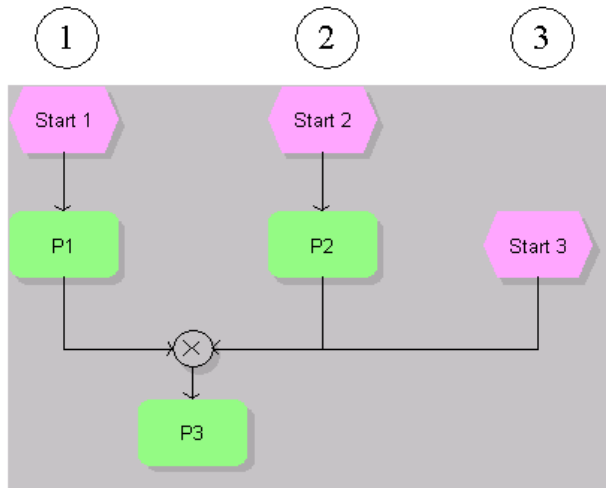


Figure 3: Processes with Several Start Events.

This rule also applies to non-disjunctive start events.

Process Branches and Joins

There are allowed combinations of operators in process branches and joins which can directly be processed by ArcStyler. In addition, there are combinations which are allowed but which require further information. Finally, there are combinations which are not allowed because they are meaningless or will result in conflicts.

ArcStyler transforms AND/AND combinations into UML Fork/Join combinations and XOR/XOR combinations into Decision/Junction combinations.

Additional information:

- Branch:

In most cases, the OR branch issue is a runtime problem that is anticipated during the creation of the model and represented in the Description/Definition of the OR element.

ArcStyler transforms the OR branch into a conditioned split in UML.

- Joins:

In UML, there are two different ways of handling joins. This must be accounted for by the ARIS modelers. After the transformation by ArcStyler of the eEPC into a UML model, **Wait for all active** (WfAA) at an OR join means that at all process branches initialized at the split are waited for at the join.




First come (FC) at an OR join means that the first process branch initialized at the split is waited for at the join. All other active branches are terminated when they reach the join. In the ARIS eEPCs, WfAA and FC are selectable values in a free attribute (for the OR operator). This attribute is referred to as *Join Semantic*.

More complex rules can be entered in the form of free text in the attribute Description/Definition of the OR joiner.

The XOR join corresponds to First Come semantics.

Other scenarios such as XOR-AND as well as AND-XOR combinations are not allowed.

Figure 4 summarizes the possible combinations:

Join \ Branch	XOR 	OR 	AND 
XOR (FC)	OK	OK <small>(corresponds to OR(FC) Join)</small>	not allowed
OR (WfAA)	OK <small>(corresponds to XOR Join)</small>	OK	OK <small>(corresponds to UND Join)</small>
OR (FC)	OK <small>(corresponds to XOR Join)</small>	OK <small>(corresponds to XOR (FC) Join)</small>	OK <small>(corresponds to XOR Join)</small>
Rule operator	OK	OK	OK
AND	not allowed	not allowed	OK

FC: First Come WfAA: Wait for all active

Figure 4: Semantics of the Operator Combinations.

Modeling the Correct Correspondence of Branches and Joins

In a number of situations, ambiguities may arise as to the correspondence of a join and the appropriate branch. While this may not be of concern to the ARIS modeler, it will cause problems

in the runtime system. Therefore, it is important to identify the corresponding branch/join combinations.

The semantics of join operators depend on the corresponding branch (cf. Figure 4). Problem: Which one is the corresponding branch?

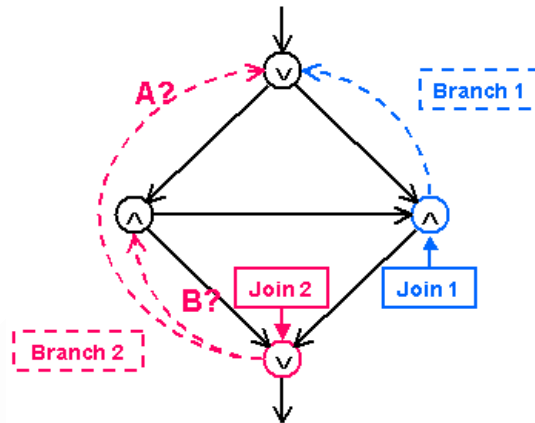


Figure 5: Correspondence of Branches and Joins.

If the semantics of the Join operator depend on the corresponding branch and if this branch is not uniquely identifiable, the ARIS modeler must specify the corresponding branch.

This can be done by identical texts string in the corresponding **Full** name.

Problematic Cycles

The following questions arise in connection with the implementation of cycles:

When is a process terminated, i.e. when are all functions (concurrency) terminated?

Which cyclical process runs lead to problems?

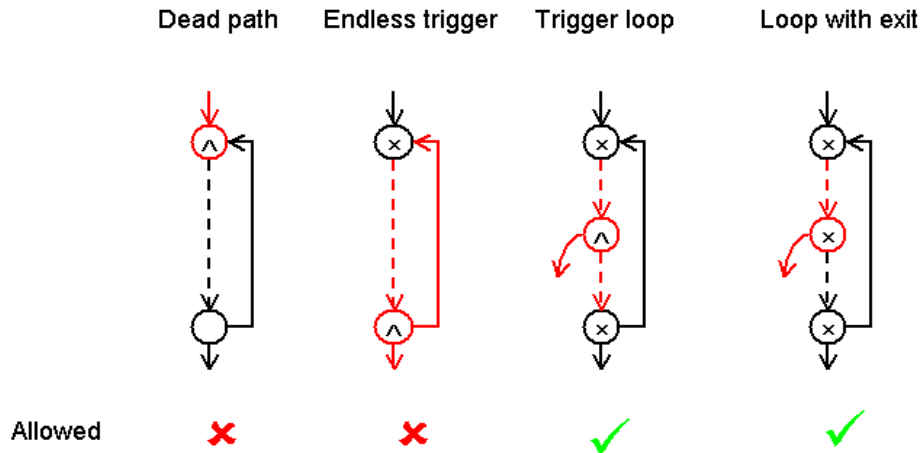


Figure 6: Cycles.

Four cycle types have been identified. The **loop with exit** and **trigger loop** are allowed. Future releases will provide verifiers for the identification of **dead paths** and **endless loops** so that they can be fixed by the modeler. This list may not be complete since, as of the date of creation of this modeling style, there may be unknown problem cases.

Concurrency

Concurrency means that after a branch there are several instances (one per branch) of a process, as opposed to the single instance prior to the branch. In the runtime environment, these concurrent processes must be synchronized so that the complete process can be properly terminated and no **dead processes** exist in the system.

The following rule applies:

A process is terminated when a end event has occurred for each activity (including concurrent activities). This means that all (partial) control flows have reached a end event.

Usually, concurrency is not modeled on purpose. In the following situations, concurrency will cause problems.

Exit or bypass

Entry

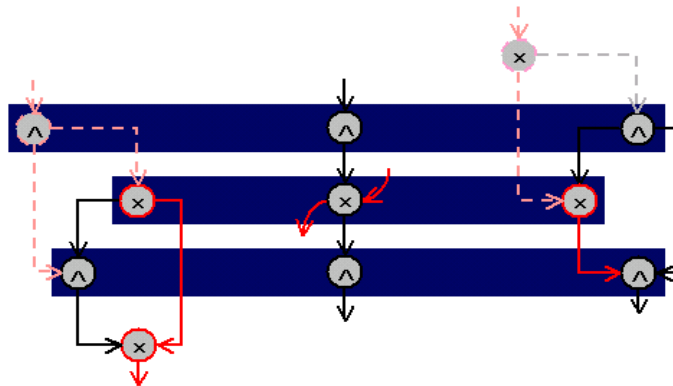


Figure 7: Terminating Concurrency.

In the case of an Exit/Bypass, the **AND** in the path coming from the **XOR** will never be reached. In the case of an Entry, the process is blocked at the bottom **AND**. The common pattern is shown in the center: an **XOR** between two **AND**s which is either the source or the target of an additional control flow.

Verifiers for the described cases and for further problems will be made available on an ongoing basis.

Verifiers

Verifiers are programs which check models for their compliance with the rules of the modeling style. It is possible to check models created in accordance with the modeling style and models not created in accordance with the modeling style.

The verifiers can be implemented in different versions with various degrees of coverage and complexity:

The model is checked and the system displays the spots where the violations of rules in the model were detected. The modeling style is used by the modelers for manual fixing of the problems.

The model is checked and the system displays at which spots the model violates which rules.

The model is checked and the system displays suggestions as to fixing the detected problems. If the model was not created according to the modeling style, this procedure resembles a Refinement Assistant which supports the modeler in converting a “free” model into a model complying with the modeling style.

The **maximum functionality** verifier would be a type of Wizard that subjects the model to real-time verification and does not even allow violations of rules.

The verifiers will be made available in upcoming releases on an ongoing basis.

Model Element Mapping (eEPC → UML)

ARIS functions are mapped to Activities. Connection-Event-Connection sequences are mapped as transitions, the event name is turned into the Guard condition of the transition.

Organizational units are mapped to packages with the stereotype **Organizational Unit**.

Positions are mapped to classes with the stereotype **Role**. In addition, Object Flow States (stereotype **Actor**) are created for Positions.

The Object States are mapped to Object Flow States (UML).

Table 1 provides a mapping overview.

Model Element and Attribute Mapping

Table 1 : Model Element and Attribute Mapping.

ARIS	ArcStyler/UML
Component group	Package
eEPC	Class with stereotype Process
Function	Activities, class with stereotype Process
Name	
Description/Definition	
Event	Guard
Name	
Description/Definition	
Operators	Join, Fork, Decision, Junction
Name	
Description/Definition	
Join Semantic	
Object State (Product/Service)	Object Flow State (UML)
Name	
Description/Definition	
Class	Class
Name	
Description/Definition	

Table 1 : Model Element and Attribute Mapping.

ARIS	ArcStyler/UML
Attribute	Attribute of a class
Name	
Description/Definition	
Cluster	Class
Name	
Description/Definition	
Entity type	Class
Name	
Description/Definition	
Organizational Unit	Package with the stereotype Organizational Unit
Name	
Description/Definition	
Organizational Unit Type	Package with the stereotype Organizational Unit Type
Name	
Description/Definition	
Position	Class with the stereotype Role and Object Flow State with the stereotype Actor
Name	
Description/Definition	
Connection	Transition/Association

Multilingual Models

.....

If you work with multilingual models, the model elements are only transformed in the database language (set via View → Options).

