How to model alternative (EXOR) Solution Building Blocks and aggregation (AND) Solution Building Blocks in a Solution model

Guidelines

Disclaimer

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

Designing solution architectures involves precisely modelling alternative (EXOR) and aggregation (AND) Solution Building Blocks. These elements are crucial for establishing relationships and constraints within the architecture, ensuring a coherent implementation aligned with business objectives.

In this document, the reader will explore how to effectively model alternative and aggregation building blocks in a solution model. The document starts by providing context on the importance of architectural constraints and their role in solution design. Then, it examines best practices and recommended approaches for expressing these constraints within the ArchiMate framework.

Architectural constraints play a pivotal role in ensuring that the solution architecture aligns with business objectives and meets stakeholder requirements. Understanding and effectively modelling these constraints is essential for creating clear and coherent architectural models.

By understanding the context and key considerations related to modelling building blocks in a solution architecture, architects will be better equipped to make informed decisions and create clear, coherent architectural models that support business objectives.

2. Fundamentals of Alternative (EXOR) and Aggregation (AND) Solution Building Blocks

This section explains in detail the fundamental concepts behind alternative and aggregation building blocks in solution architecture:

- **Definition of Alternative (EXOR) Building Blocks:** Alternative building blocks represent mutually exclusive options within an architecture. These elements allow modelling scenarios where only one option can be selected at a given time. For example, in a sales management system, there might be an option for online or offline order processing, but not both simultaneously.
- **Definition of Aggregation (AND) Building Blocks:** Aggregation building blocks represent options that can coexist within an architecture. These elements allow modelling scenarios where multiple options must be selected and work together in tandem. For example, in an inventory management system, there could be options for real-time inventory tracking and automatic scheduling of deliveries, both functioning simultaneously.

This document provides practical examples illustrating how alternative and aggregation building blocks are used in different solution architecture contexts. These examples will help clarify the concepts and provide a stronger understanding of their application in practice.

3. Introduction to Modelling Constraints

In this section, it is explored the foundational aspects of modelling constraints, focusing on the modelling of alternative (EXOR) and aggregation (AND) building blocks.

Modelling EXOR/AND building blocks revolves around defining constraints within the architecture. These constraints serve to establish rules and conditions governing the relationships between various elements. In UML (Unified Modelling Language), architects use a specific constraint relationship to articulate these constraints. This relationship allows architects to express a wide range of rules using the Object Constraint Language (OCL), a formal language designed for precisely defining constraints.

The Object Constraint Language (OCL) is a powerful tool that enables architects to specify complex constraints with precision. It includes logical operators such as XOR (exclusive OR) and AND, which allow architects to define constraints involving mutually exclusive or simultaneous conditions, respectively.

For instance, consider a scenario where a "Subject" entity in a system can only be associated with either a "Person" or a "Company," but not both simultaneously. This constraint can be expressed using an XOR constraint, indicating

that the relationship between "Subject" and "Person" is mutually exclusive with the relationship between "Subject" and "Company."

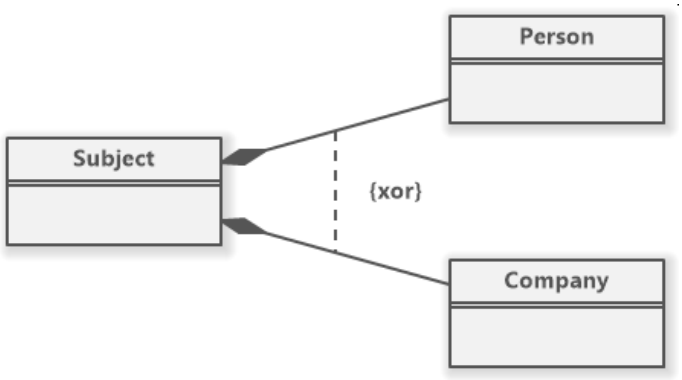


Figure 1. EXOR Modelling Constraint

By exploiting the capabilities of UML and the Object Constraint Language, architects can accurately model EXOR and AND building blocks within the architecture, ensuring that the relationships between elements adhere to the specified constraints.

Proposed Guidelines:

- When modelling building blocks in ArchiMate, it is crucial to consider associated constraints.
- Constraints can be expressed using the UML constraint relationship or by employing the Object Constraint Language (OCL).
- Within the ArchiMate context, constraints can be modelled using the association relationship, although this may not be the standard practice.
- The use of the ArchiMate constraint element is recommended for a more precise and standardized representation of constraints.
- When defining constraints, it is important to ensure they are clear and understandable for modellers.

4. Approach in ArchiMate

This section proposes the approach to modelling constraints within the ArchiMate framework. It presents a simplistic approach to representing constraints, focusing on the utilization of the association relationship and comparing it with the convention in UML.

Coming to ArchiMate and approaching this simplistically, architects can attempt to model constraints using the most relaxed “association” relationship, given that there is no constraint relationship. Unlike UML, ArchiMate does not have a dedicated constraint relationship. Therefore, architects often leverage existing relationships to express constraints, albeit with certain limitations. Then it could be added a label on this association matching the UML approach, for example:

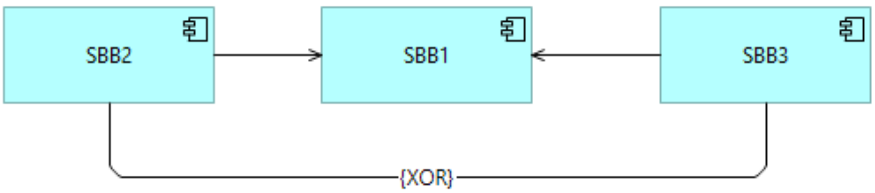


Figure 2. XOR - Labelling the association relationship

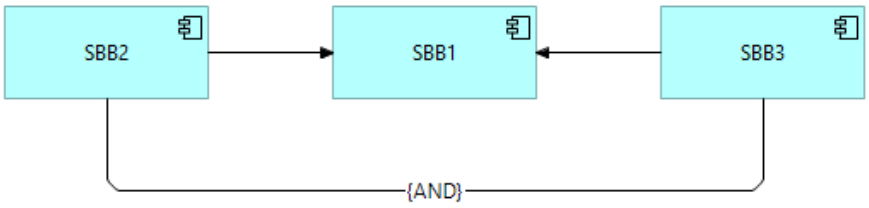


Figure 3. AND - Labelling the association relationship

In this approach, architects use the association relationship between elements to represent constraints in ArchiMate. By adding labels or annotations to these associations, architects can convey the constraints in a structured manner.

5. USE OF ARCHIMATE CONSTRAINT ELEMENT

This section approaches the usage of the ArchiMate constraint element as an alternative method for modelling constraints within the EIRA and eGovERA frameworks. Unlike the traditional approach of relying solely on the association relationship, the ArchiMate constraint element offers a more robust and standardized way of representing constraints within architectural models.

Expanding upon this alternative approach, architects have the flexibility to extend its usage beyond simple constraints. This includes modelling "AND" and other Boolean constraints, as well as incorporating Object Constraint Language (OCL) constraints. By doing so, architects can achieve a level of flexibility and expressiveness akin to the conventions observed in UML modelling.

However, it is essential to recognise that this extension constitutes a departure from the standard representation of constraints in ArchiMate. While resembling the approach in UML, where logical operators like XOR are attached to elements, it remains a convention rather than the norm within the ArchiMate framework.

To align with the established standards and ensure clarity and precision in constraint representation, ArchiMate provides the dedicated Constraint element. This element serves the specific purpose of modelling constraints within architectural models. Utilizing the Constraint element, architects can precisely define constraints and maintain consistency across the architectural framework.

Representing the aforementioned constraints using this element is illustrated below:

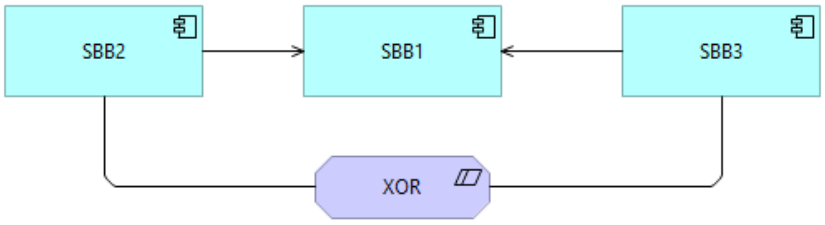


Figure 4. XOR - Archimate constraint element

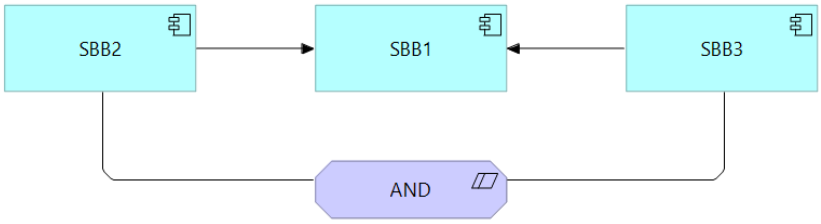


Figure 5. AND - Archimate constraint element

Overall, while the extension of the association relationship approach may offer similarities to UML conventions, the adoption of the ArchiMate constraint element ensures adherence to standard practices and provides a more structured and standardized representation of constraints within architectural models.

5.1. Clarifying Constraint Labels

The strategy of clarifying constraint labels represents a departure from EIRA-specific conventions, bringing it more in line with the foundational principles of ArchiMate. While this approach offers several advantages, such as enhanced clarity and alignment with industry standards, it also introduces a potential risk.

One potential risk is the possibility of conflating real-world concepts modelled using the constraint element with constraints related to modelling options and guidelines. This could lead to confusion and misinterpretation within the architectural model, potentially impacting the overall effectiveness of the design.

To mitigate this risk, a recommended approach is to adopt the label notation commonly used in UML modelling. This involves encapsulating the constraint label within curly brackets, providing a clear visual distinction between constraints and other architectural elements.

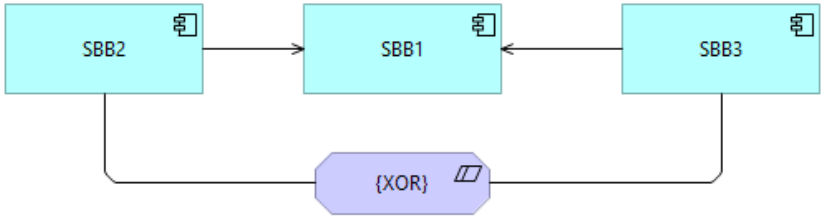


Figure 6. XOR - Constraint labels

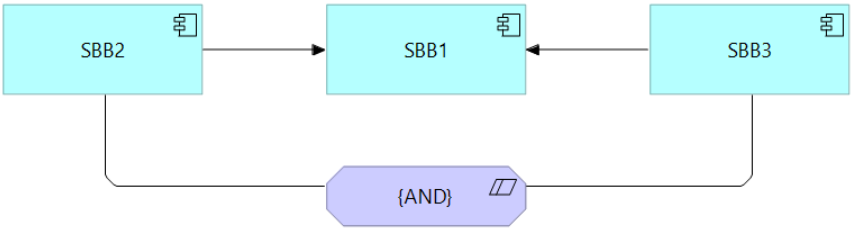


Figure 7. AND - Constraint labels

By implementing this notation convention, architects can effectively communicate the purpose and intent of each constraint, minimizing ambiguity and ensuring the clarity of the architectural representation. This approach promotes consistency and standardization within the architectural model, facilitating better comprehension and interpretation by stakeholders.

In summary, while clarifying constraint labels offers benefits in terms of alignment with core ArchiMate principles, architects must remain vigilant in managing the potential risks associated with this approach. By adhering to established notation conventions and maintaining clear communication, architects can effectively leverage constraint labelling to enhance the overall quality and clarity of architectural models.

5.2. Enhanced Expressiveness with Constraint Element

The use of the ArchiMate constraint element enriches the expressiveness of architectural models, providing clarity and precision in conveying constraints. By employing this element, architects indicate to modellers that the constraint's value should be understood as Object Constraint Language (OCL), ensuring consistency and comprehension throughout the architectural depiction.

Adopting the ArchiMate constraint element offers several advantages over alternative approaches, such as using association relationships. Notably, it provides greater expressiveness, allowing architects to articulate constraints with more detail and nuance. Unlike the association approach, which may limit the scope of constraint representation, the constraint element offers a flexible framework for expressing complex constraints effectively.

Moreover, the ArchiMate constraint element enables architects to attach constraints directly to associations, broadening the possibilities for constraint modelling. This capability empowers architects to define constraints about specific associations, enhancing the precision and granularity of constraint representation within the architectural model.

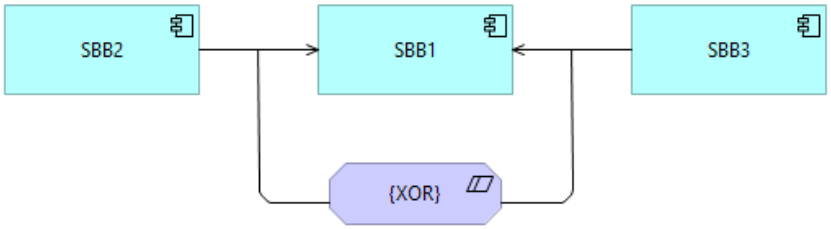


Figure 8. XOR - Expressiveness with constraint element

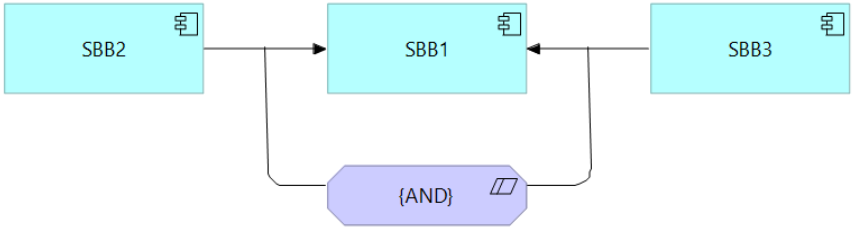


Figure 9. AND - Expressiveness with constraint element

Overall, harnessing the ArchiMate constraint element enhances the expressiveness of architectural models, enabling architects to articulate constraints with clarity and precision. Through the use of this element, architects can effectively communicate complex constraints, fostering a deeper understanding of architectural constraints among stakeholders.

6. Comparison of Models

When comparing different approaches to modelling constraints in ArchiMate, it is essential to understand how each method influences the conceptual representation within the architectural model.

In the context of the discussion, consider two models: one employing the association approach and the other using the ArchiMate constraint element.

The first model, using the association approach, stipulates that either SBB2 or SBB3 can be defined in the model, but not both concurrently. This implies a mutually exclusive relationship between SBB2 and SBB3, restricting the possibility of their simultaneous existence within the architectural model.

In contrast, the second model, leveraging the ArchiMate constraint element, allows for a more nuanced representation. Here, SBB2 and SBB3 can coexist within the model, but a constraint specifies that SBB1 can only be associated with one of them, not both concurrently. This model introduces a finer distinction, enabling SBB2 and SBB3 to function together while maintaining constraints on their relationship with SBB1.

One notable limitation of the association approach is its inability to represent relationships between relationships directly. ArchiMate mandates that relationships must connect elements directly, without intermediary relationships. As a result, the association approach may encounter difficulties in expressing complex constraints involving multiple elements and relationships.

Despite the perceived verbosity of adding a Constraint element, its usage offers advantages in handling complex cases. Architects can employ groupings to achieve the same semantics, providing a structured and comprehensible representation of constraints within the architectural model.

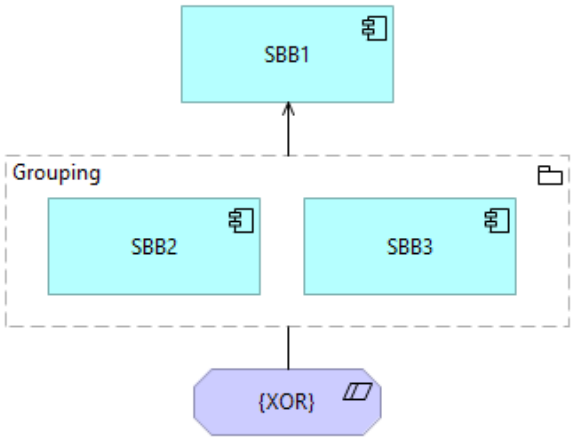


Figure 10. XOR - Usage of groupings for comparison of models

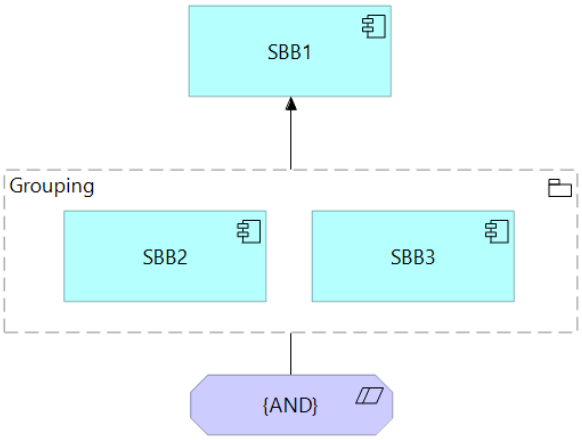


Figure 11. AND - Usage of groupings for comparison of models

In summary, while both approaches have their merits, the ArchiMate constraint element offers a more refined and expressive means of modelling constraints, particularly in scenarios involving intricate relationships between architectural elements.

7. Supporting Other Constraints

When considering the support for different types of constraints, it is essential to address the immediate requirements for representing XOR and AND constraints. However, it is also crucial to acknowledge the potential for accommodating a broader range of constraints within the architectural model.

By encapsulating constraints within curly brackets and specifying them as Object Constraint Language (OCL) expressions, architects can offer flexibility for modelling various types of constraints. This approach allows modellers to define constraints based on specific project needs, enabling them to express complex logical conditions or business rules effectively.

While ArchiMate primarily focuses on graphical representation rather than formal expressions, incorporating OCL expressions within constraints can enhance the modelling capabilities of the framework. Despite this flexibility, it is important to maintain clarity and relevance in constraint definitions, ensuring that they contribute meaningfully to the architectural representation.

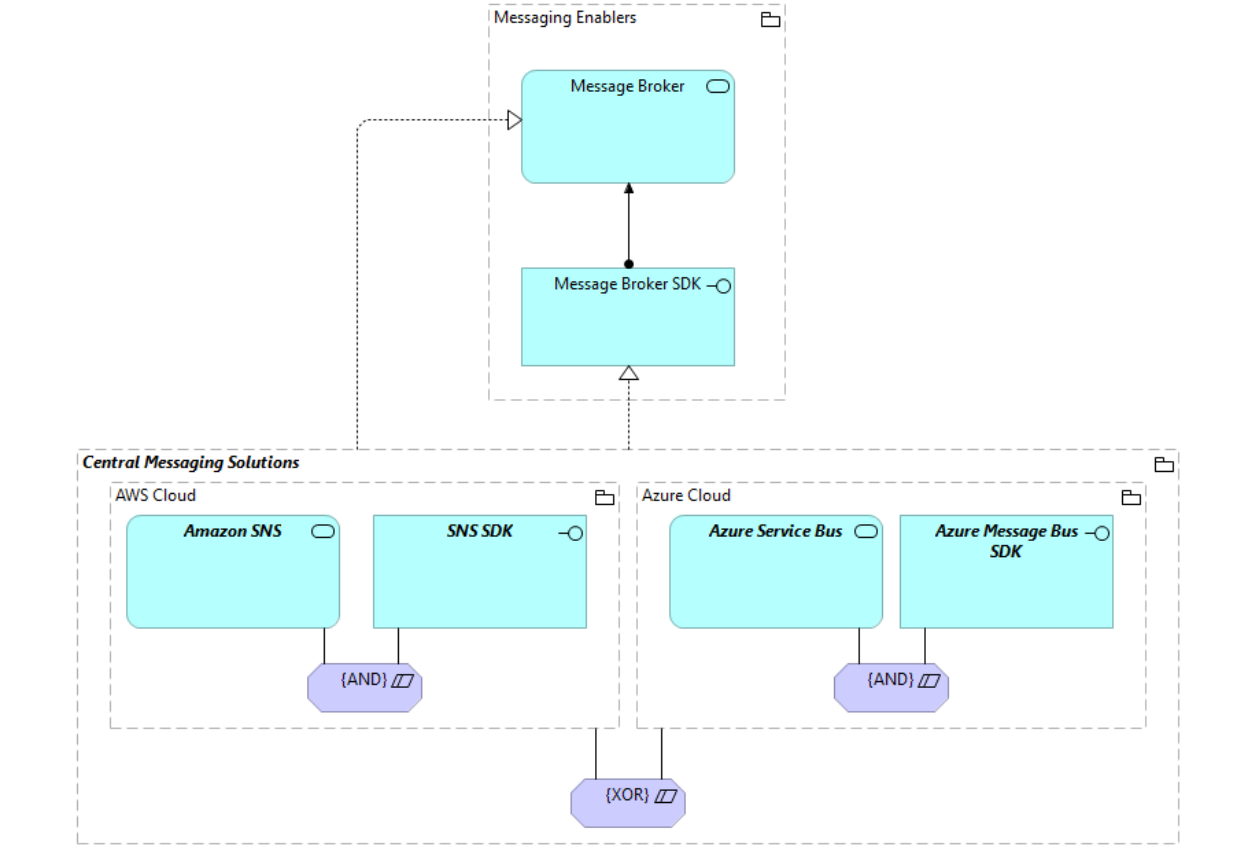
Architects should encourage modellers to use curly brackets to denote OCL expressions when defining constraints, providing a clear indication of their nature and purpose within the architectural model. Additionally, abstract constraints can be represented simply by omitting curly brackets, allowing for more straightforward modelling of conceptual constraints such as Memoranda of Understanding (MoUs).

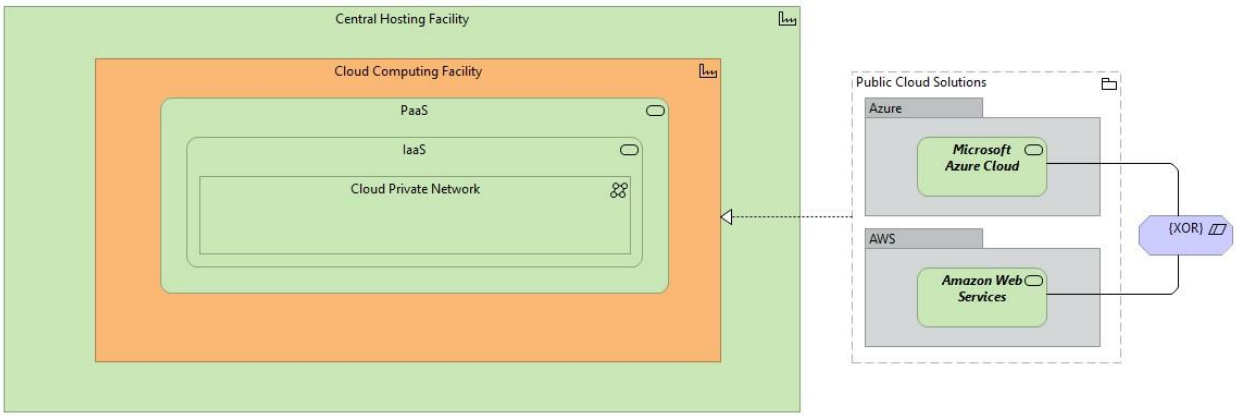
It is essential to establish guidelines for how constraints are supported and validated within the architectural model. The EIRA validator plays a critical role in detecting and enforcing XOR and AND constraints using existing techniques. However, it is important to note that validation beyond a predefined set of OCL constraints may not be supported by the validator.

By providing guidance on the use of curly brackets for OCL expressions and clarifying the role of the EIRA validator in constraint validation, architects can ensure consistency and effectiveness in modelling constraints within the architectural framework. This approach empowers modellers to express complex constraints while maintaining alignment with architectural standards and objectives.

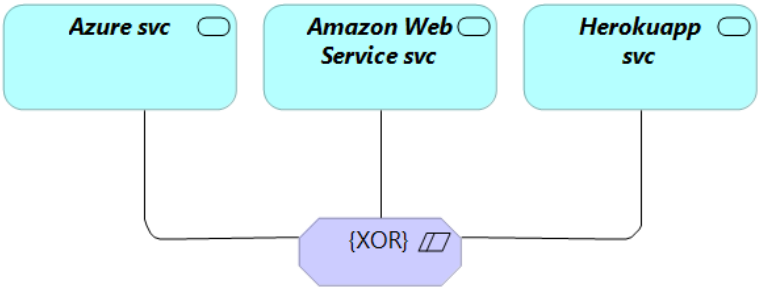
ANNEX I – REAL EXAMPLE

The following figures represent a real example using the combination EXOR and AND elements:





The following figures represent a real example using the EXOR:



The following figures represent a real example using the AND: