



**WP 8.1.4 – DEFINE
METHODOLOGY FOR VALIDATION
WITHIN OATA**

**Architecture Tactics Assessment
Process**

Document Identifier:	OATA-P2-D8.1.4-01
Edition:	2.0
Edition Date:	24-04-2006

DOCUMENT CHARACTERISTICS

TITLE			
Architecture Tactics Assessment Process			
EATM Infocentre Reference:			
Document Identifier:	OATA-P2-D8.1.4-01	Edition:	2.0
Contractual Ref:	D5c	Version Date:	24-04-2006
Contractual ID:			
Abstract			
<p>This document presents a methodology for the assessment of architectural tactics and patterns made during the development of OATA logical architecture.</p> <p>Through architecture inspection, the validation team identifies maintainability tactics, interoperability patterns and system interoperability. The result is assessed and recommendations are proposed in order to improve and increase the knowledge of the logical architecture.</p>			
Keywords			
Validation Pattern	Logical Architecture Interoperability	Method	Tactics
Contact Person(s)		Tel	Unit
Prepared by:	SWEDAVIA,ISDEFE,COMBITECH		
Issued by:	Hans Wagemans, WP8.1 Manager	+32 2 729 3334	DAS/ESC

STATUS, AUDIENCE AND ACCESSIBILITY			
Status		Intended for	Accessible via
In progress	<input type="checkbox"/>	General Public	<input type="checkbox"/> Intranet <input checked="" type="checkbox"/>
Internal Draft	<input type="checkbox"/>	EATM Stakeholders	<input checked="" type="checkbox"/> Extranet <input type="checkbox"/>
Working Draft	<input type="checkbox"/>	Restricted Audience	<input type="checkbox"/> Configuration Manager <input type="checkbox"/>
Proposed Issue	<input checked="" type="checkbox"/>	<i>Printed & electronic copies of the document can be obtained from the EATM Infocentre or from the OATA PSO</i>	
Released Issue	<input type="checkbox"/>		

ELECTRONIC SOURCE		
Path:	http://pollux.mis.eurocontrol.be/iw/cci/meta/no-injection/iw-mount/default/main/template_web/eatm/valfor/WORKAREA/work/web/gallery/content/public/zz_tst_borkenhu_1_OATA-P2-D8.1.4-01 DMVO Architecture Tactics Assessment Process.doc	
File Name:	zz_tst_borkenhu_1_OATA-P2-D8.1.4-01 DMVO Architecture Tactics Assessment Process	
Host System:	Software Application	Size:
Windows XP:	Microsoft Word 10.0	1142 Kb

EATM Infocentre EUROCONTROL Headquarters 96 Rue de la Fusée, B-1130 BRUSSELS Tel: +32 (0)2 729 51 51 Fax: +32 (0)2 729 99 84 E-mail: eatm.infocentre@eurocontrol.int	OATA Project Support Office (PSO) EUROCONTROL Headquarters 96 Rue de la Fusée, B-1130 BRUSSELS Tel: +32 (0)2 729 50 40 E-mail: oata.pso@eurocontrol.int
--	--

DOCUMENT APPROVAL

The following table identifies all management authorities who have successively approved the present issue of this document.

AUTHORITY	NAME AND SIGNATURE	DATE
Contractor	SWEDAVIA	12-04-2006
Work Package Manager	Hans Wagemans	03-05-2006
Internal Review Board		
Technical Review Group		
Project Manager		

DOCUMENT CONTROL

Copyright notice

© 2007 European Organisation for the Safety of Air Navigation (EUROCONTROL).
 All rights reserved.
 "Member States of the Organisation are entitled to use and reproduce this document for internal and non-commercial purpose under their vested tasks. Any disclosure to third parties shall be subject to prior written permission of EUROCONTROL".

DOCUMENT CHANGE RECORD

The following table records the complete history of the successive editions of the present document.

Edition Number	Edition Date	Reason for change	Pages affected
0.A	4 June 2004	Creation	all
0.B	7 Feb 2005 Internal Version not delivered	Reordering	all
0.C	10 Mar 2005	Updates after review by OATA (4/3/2005)	all
0.D	6 April 2005	Updates after review	all
1.0	13 April 2005	IRB review comments	all
1.1	5 April 2006	Update after the Pilot Validation including internal review.	all

		The major changes are the movement of elaborate Inspection Guidelines from the execution activities to separate activities in the preparation process and an activity for the repository. Two new roles are introduced; Documentation Expert and Repository Expert. Added information and recommendations gained in the Pilot Validation.	
2.0	24 April 2006	Update after IRB 12-04-2006	

TABLE OF CONTENTS

DOCUMENT CHARACTERISTICS II

DOCUMENT APPROVAL..... III

DOCUMENT CONTROL..... III

DOCUMENT CHANGE RECORD III

TABLE OF CONTENTS..... V

LIST OF FIGURES..... VI

EXECUTIVE SUMMARY..... 7

1 INTRODUCTION..... 8

1.1 DOCUMENT BACKGROUND..... 8

1.2 DOCUMENT STRUCTURE 8

1.3 READING GUIDELINES..... 9

1.4 DEFINITIONS AND TERMS..... 9

1.5 BIBLIOGRAPHY..... 12

2 METHODOLOGY OVERVIEW..... 13

2.1 INTRODUCTION 13

2.2 OVERVIEW OF TYPICAL ARCHITECTURE ASSUMPTIONS 13

2.2.1 Architecture Tactics 13

2.2.2 Architecture and Interoperability..... 14

2.3 CONDUCTING AN INSPECTION OF AN ARCHITECTURE 16

2.3.1 Introduction 16

2.3.2 Elaboration of Inspection Guidelines 16

2.4 INSPECTION REVIEW 16

3 PROCESSES 17

3.1 INTRODUCTION 17

3.1.1 Required Roles 21

3.2 PREPARATION..... 22

3.2.1 Introduction 22

3.2.2 Required Resources 24

3.2.3 Activity: Select the OATA Architecture 24

3.2.4 Activity: Plan the Validation Cycle 25

3.2.5 Activity: Prepare the repository..... 26

3.2.6 Activity: Elaborate Inspection Guidelines for Maintainability 27

3.2.7 Activity: Elaborate Inspection Guidelines for Interoperability 31

3.3 EXECUTION 32

3.3.1 Introduction 32

3.3.2 Required Resources 35

3.3.3 Activity: Identification of Maintainability Tactics..... 35

3.3.4 Activity: Identification of Interoperability Patterns..... 36

3.3.5 Activity: Identification of System Interoperability 37

3.4 ANALYSIS AND REPORTING..... 39

3.4.1 Introduction 39

3.4.2 Required Resources 41

3.4.3 Activity: Propose Recommendations..... 41

3.4.4 Activity: Write the Validation Report(s)..... 42

4	APPENDIX A: TACTICS AND PATTERNS	44
4.1	INTRODUCTION	44
4.1.1	Architecture Tactics	44
4.1.2	Architectural Patterns	44
4.1.3	Architecture and Interoperability	44
4.1.4	Data Interoperability	46
4.1.5	Architecture Styles for Addressing Interoperability.....	47
4.2	MAINTAINABILITY TACTICS.....	49
4.2.1	Tactics for Localizing Expected Modifications.....	49
4.2.2	Tactics for Restricting the Visibility of Responsibilities.....	50
4.2.3	Tactics for Preventing the Ripple Effect	50
4.3	EFFICIENCY CONTROL TACTICS	51
4.3.1	Efficiency Requirements Tactic	51
4.3.2	Instrumenting Tactic	52
4.4	INDEPENDENT TACTICS	52
4.4.1	Centring Tactic.....	52
4.4.2	Fixing-Point Tactic	52
4.4.3	Locality Tactic	53
4.4.4	Processing Versus Frequency Tactic.....	53
4.5	SYNERGISTIC TACTICS	53
4.5.1	Shared Resources Tactic	53
4.5.2	Parallel Processing Tactic	54
4.5.3	Spread-The-Load Tactic.....	54
4.6	PATTERNS.....	54
4.6.1	Translator Patterns	55
4.6.2	Controller Patterns.....	56
4.6.3	Extender Patterns.....	57
5	APPENDIX B: SEMANTIC INTEROPERABILITY	58
5.1	INTRODUCTION	58
5.2	ELABORATION OF AN ONTOLOGY	58

LIST OF FIGURES

Figure 1:	Summary of the Architecture Validation Process	8
Figure 2:	Overall Flow of the Architecting Stage	14
Figure 3:	Process Overview	18
Figure 4:	Resource Overview	19
Figure 5:	Information Flow Overview	20
Figure 6:	Preparation.....	23
Figure 7:	Execution	34
Figure 8:	Analysis and Reporting.....	40
Figure 9:	1Architectural Styles addressing interoperability.....	47
Figure 10:	Interoperability patterns Taxonomy.....	55
Figure 11:	Translator Patterns Schema	56
Figure 12:	Controller Patterns Schema.....	56
Figure 13:	Extender Patterns Schema.....	57

EXECUTIVE SUMMARY

The purpose of this document is to present a methodology to identify, understand and assess implicit and explicit assumptions, which were made during the development of the OATA logical architecture. The methodology will try to make these assumptions explicit by identifying tactics and patterns that represent important properties regarding the development of an architecture: maintainability, general interoperability and system interoperability.

Tactics and patterns in the architecture are mainly used for two reasons: to follow established practice, and to simplify a difficult analysis/design problem. In most situations tactics and patterns are needed to ensure a correct and coherent architecture.

Assumptions will be identified through architecture inspections performed by a team composed of both OATA architecture experts and architecture validation experts.

Identified assumptions will be analysed and assessed and finally, the validation team will propose recommendations how OATA logical architecture could be improved.

1 INTRODUCTION

1.1 Document Background

The purpose of the validation in OATA is to increase confidence in the architecture being developed, and to ensure that it meets Stakeholders' expectations and quality criteria.

The validation during the early development life cycle will be performed around four areas: Architecture Compliance with User Needs, Quality of the Architecture Structure, Compliance with Non-Functional Requirements, and Architecture Tactics assessment.

This document is dealing with the Architecture tactics assessment process and presents a methodology to identify, understand and assess implicit and explicit assumptions, which were made during the development of the OATA logical architecture. The methodology will look for tactics and patterns in the logical model that represent important properties regarding the development of an architecture:

- Maintainability
- General Interoperability
- System Interoperability

Figure 1 shows the four areas of the OATA validation; the area that this document is concerned with is expanded.

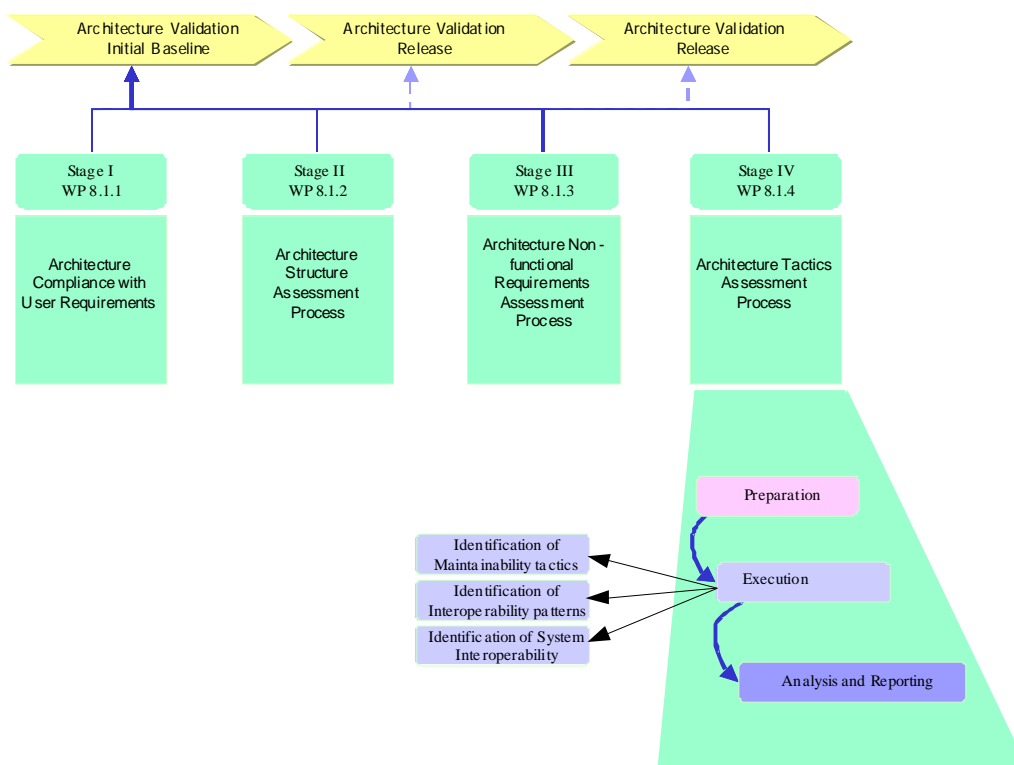


Figure 1: Summary of the Architecture Validation Process

1.2 Document structure

This document has the following main chapters:

- **Introduction**

The introduction contains description of the background of this document, definitions and terms and a list of references.

- **Methodology Overview**

The methodology overview contains a high level description of the validation methodology to be used in the validation.

- **Processes**

The processes chapter details each part of the validation process and describes the activities that must be undertaken during a validation cycle.

There are also two appendices included in this document that gives a more detailed description of elements of the validation methodology:

- Tactics and Patterns (Appendix A)
- Semantic Interoperability (Appendix B)

1.3 Reading guidelines

The processes in this methodology are presented as process diagrams using standard UML Business Modelling syntax, [Eriksson, Penker 00].

The process diagram contains information about:

- **Processes and Activities**

A process is a set of related activities and they are drawn in the centre/middle of the process diagrams with solid arrows linking processes/activities that follow one another.

- **Resources**

Resources are the objects within the business, such as people, material, information, and products, which are used or produced in the business. Resources are manipulated (used, consumed, refined, or produced) through processes.

Controlling resources control or run a process. These resources are drawn above a process with a dashed line from the resource to the process.

Supporting resources participate in a process and are not refined or consumed. These resources are drawn below a process with a dashed line from the resource to the process.

Input information resources are placed to the left of the process and connected to the process with a dashed line.

Output information resources are placed to the right of the process and connected to the process with a dashed line.

- **Activity synchronization**

Activities can be performed in parallel. A need for synchronization is shown in the process diagram by using a vertical synchronization bar. The activities after the synchronization bar must wait for all the activities before the bar to complete before they can start.

A detailed process diagram contains activities and their relationships.

For more detailed information see [Eriksson, Penker 00].

1.4 Definitions and Terms

This section presents some of the basic terms that are needed to understand this document. More acronyms can be found on the web.

http://www.eurocontrol.int/oca/public/site_preferences/display_glossary_list.html

Architecture

Architecture is a hierarchical description for the design of a system. In many cases it describes how it will be developed, evolved, and operated. Architecture provides the underlying blueprint for the more detailed design and implementation decisions about the components of a system.

Architecture tactic

An architecture tactic is a means of satisfying a quality characteristic by balancing the aspects of quality attributes through architectural design decisions.

Architecture pattern

Architecture pattern show how to organise a set of classes in order to achieve a specific quality characteristic from the architecture. More formally, Architectural Pattern “expresses a fundamental structural organisation schema for software systems”. They provide a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organising the relationships between them.

Architecture style

An architectural style is a family of architectures constrained by component/connector vocabulary, topology, semantic constraints and analysis methods supported. Styles can be different because they use different vocabularies, topologies or constraints. The vocabulary of component/connector is frequently the principal aspect characterising the style.

[Shaw 95]

Maintainability

Quality characteristics related effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functional specifications.

The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.

Maintainability has the following subcharacteristics:

- **Analysability:** The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.
- **Changeability:** The capability of the software product to enable a specified modification to be implemented.
- **Stability:** The capability of the software product to avoid unexpected effects from modifications of the software.
- **Testability:** The capability of the software product to enable modified software to be validated.

[ISO 9126]

Validation Aim

A sentence, that describes Stakeholders' expectations and needs, which shall be reflected in OATA.

Validation Objective

A statement of what the project team will work towards to achieve the aim. A formulation of the validation aims in measurable factors.

Quality

The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs.

[ISO 9126]

Note: An entity can be a product, service or process

Quality Attribute

Quality Attribute is a measurable physical or abstract property of an entity (product or service).

[ISO 9126]

Quality Model

Set of characteristics and relationships between them, which provide the basis for specifying quality requirements and evaluating quality.

[ISO 9126]

Quality Characteristic

Inherent characteristic of a product, process or system related to a requirement.

[ISO 9000:2000]

Role

A required resource with a specific responsibility and knowledge suitable for validation. A role may be assigned to one or several resources (Validation Team Members).

UML

Unified Modelling Language, an OMG standard for visual object-oriented modelling.

Validation Team

A set of roles required for the conduct of the validation methodology. Each role may be assigned to one or several resources.

Validation Team Member

A resource which has been assigned one or more specific roles.

1.5 Bibliography

Eriksson, Penker 00	H-E Eriksson, M Penker, "Business Modeling with UML", 2000
Bass 98	Bass L.; Clements P. and Kazman R. Software Architecture in Practice. Reading, MA. Addison Wesley 1998.
Buschmann 96	Buschmann F.; Meunier R.; Rohnert, H.; Sommerlad, P.; and Stal M. Pattern Oriented Software Architecture, Volume 1: A System of Patterns. New York, John Wiley & Sons, 1996.
Davis 00	Under the Influence: How System Architectures Impede Interoperability, L. Davis, J. Payton, R. Gamble, 2nd International Workshop on Software Performance (2000)
IEEE	Institute of Electrical and Electronic Engineers
Keshav 98	R. Keshav and R. Gamble, Mathematical and Computer Sciences - University of Tulsa, Towards a Taxonomy of Architecture Integration Strategies, 3rd International Software Architecture Workshop, Nov. 1-2, 1998.
OATA Modelling Guide	Eurocontrol, OATA-P2-D2.5-01 OATA Modelling Guide
OATA Method Overview	Eurocontrol, H. Wagemans OATA-P2-D8.1-01 Validation Methodology Overview
OATA WP 8.1.1	Eurocontrol, H. Wagemans, OATA-P2-D8.1.1-01, WP 8.1 "Define Methodology for Validation within OATA - Architecture Compliance Assessment Process"
OATA WP 8.1.2	Eurocontrol, H. Wagemans, OATA-P2-D8.1.2-01, WP 8.1 "Define Methodology for Validation within OATA - Architecture Structural Assessment Process"
OATA WP 8.1.3	Eurocontrol, H. Wagemans, OATA-P2-D8.1.3-01, WP 8.1 "Define Methodology for Validation within OATA - Architecture Non-Functional Assessment Process"
OMG 98	Object Management Group. CORBA 2.2 Specification (OMG 98-07-01). Framingham, Ma.: Object Management Group, 1998. <URL: http://www.omg.org/library/c2indx.html >.
Shaw 95	Shaw, Mary, et al. "Abstractions for Software Architecture and Tools to Support Them." IEEE Transactions on Software Engineering 21, 6 (April 1995): 314-335
Shaw 96	Mary Shaw, David Garlan, Software architecture: perspectives on an emerging discipline, Prentice-Hall, Inc., Upper Saddle River, NJ, 1996
Visser 03	Ontology-Based information integration, U. Visser, H. Stuckenschmidt & H. Wache, IJCAI-Tutorial SP5, Acapulco Mexico, August 2003

2 METHODOLOGY OVERVIEW

2.1 Introduction

The purpose of this methodology is to identify, understand and assess implicit and explicit assumptions, which were made during the elaboration of the OATA logical architecture.

Assumptions in the architecture are mainly made for two reasons: to follow established practice, and to simplify a difficult analysis/design problem. In most situations assumptions are needed to ensure a correct and coherent architecture.

It is not an easy task to identify underlying assumptions in a logical architecture. Therefore it's important to have expertise knowledge of typical assumptions, i.e. architecture tactics and patterns. In appendix A, several tactics and patterns that can be used during the development of a logical architecture are described.

To identify the tactics and patterns, it's important to have expert knowledge of typical assumptions, therefore a validation team composed of both OATA architecture experts and architecture validation experts will identify the maintainability tactics, interoperability patterns and system interoperability aspects of the OATA architecture.

Identified tactics and patterns will be analysed and assessed and finally, the validation team will propose recommendations how OATA logical architecture could be improved.

Some tactics to be used during the development of the logical model are defined in the UML modelling guide and will/should be recognised in the validation exercises; other tactics found in the validation exercises should be further analysed and assessed for validity.

The following sections give an overview of typical architectural assumptions and architecture inspection.

2.2 Overview of Typical Architecture Assumptions

The typical assumptions (architecture tactics) that will be looked for during an architecture inspection represent important properties regarding the development of an architecture:

- Maintainability (identification of maintainability tactics)
- General Interoperability (identification of interoperability patterns)
- System Interoperability
- (Efficiency¹) (identification of efficiency control tactics)

Tactics can be made for two reasons: to follow established practices (and sometimes common sense), and to simplify a difficult analysis/design problem by applying known solutions. In most situations these tactics are needed to ensure a correct and coherent architecture.

The following paragraphs summarise the most typical tactics that can be used when elaborating a logical architecture, as well as some of the theoretical background (more details can be found in Appendix A) which is needed to understand their identification.

2.2.1 Architecture Tactics

Architecture tactics describe how a particular quality attribute is achieved through the use of architecture components. Thus tactics shows how to “address” an issue related to a quality

¹ Efficiency is included in this list but it will not be further developed in this document. The reason is that tactics exist to control efficiency (see appendix A), but efficiency is more related to physical architecture aspects and the validation method defined in this document is for assessment of the OATA logical model. Note: Some efficiency aspects are covered in WP8.1.3.

attribute. More formally, an architecture tactic is *a means of satisfying a quality attribute response measure by balancing quality aspects through architectural design decisions*.

An architectural tactic specifies how the parameters related to the quality of the architecture can be achieved -through architectural decisions- to accomplish a desired response measure.

Figure 2 shows the overall flow of the architecture development and how tactics can be applied to make some design decisions.

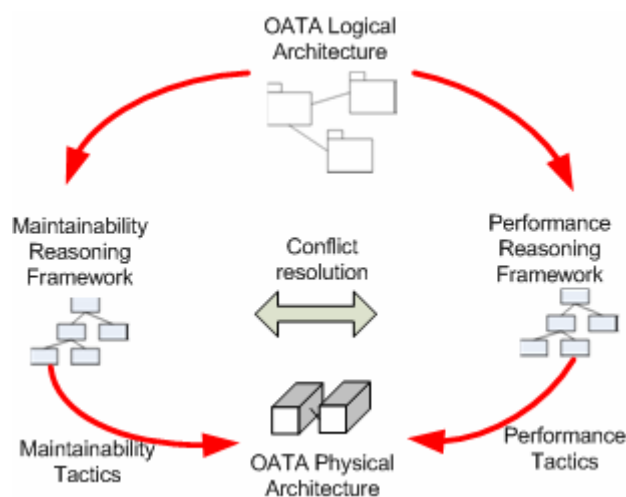


Figure 2: Overall Flow of the Architecting Stage

Maintainability tactics are considered suitable for assessment of the OATA logical architecture and will be explored further in this document (see section 4.2). The performance (efficiency) tactics are related more to implementation aspects and, therefore, will not be developed further as part of this validation activity.

2.2.2 Architecture and Interoperability

One of the most important aspects of the OATA logical architecture will be to ensure its interoperability both internally (between the different modules) and externally (both with legacy and new systems). When reasoning about architecture, it makes sense to strive for an environment based on well-defined requirements specification, common data structures, common interface requirements, and well-specified high-level information flows. Systems constructed in accordance with such an architecture are likely to be adequately interoperable. However, particularly when legacy systems are involved, these commonalities may not exist. In these cases, architectures can supply valuable guidance to isolate gaps and risks relative to interoperability.

To ensure interoperability across modules that could be built with different technologies and implemented for different environments, the negotiation and combination of properties is at the level of architectural specifications, since at this level specific implementation details can be abstracted away. Hence the OATA logical architecture will ensure that the compatibility and the interactions among modules are verified early. Thus the key to interoperability lies in the architecture style itself; to ensure that the issue is both clearly understood and explored, the following text describes the different issues related to architecture and interoperability.

2.2.2.1 Interoperability

During the Pilot Validation it was discovered that validation of interoperability as it is described in this document is not effectively applicable on the OATA Architecture. The methodology of interoperability validation is under re-construction and will be changed.

[IEEE] has several definitions of interoperability, among those are:

- The ability of two or more systems or modules to exchange information and to use the information that has been exchanged.
- The capability for two or more systems or modules to work together to do useful functions.

Verification of conformity to the standards and criteria that facilitate interoperability gives a high degree of confidence in the interoperability of systems using those standards. However, the confidence in interoperability given by conformity to one or more standards is not always sufficient and there may be need to use an interoperability assessment methodology in demonstrating interoperability between two or more systems in practice.

From the definitions that were given previously it is possible to view interoperability from three points of view:

- **System:** System characteristics are concerned with the modules as a whole and how they shape the system. Data characteristics are concerned with how data resides and moves in the system. Control characteristics, similar to data characteristics, address control issues in the system. Architectural style provides information regarding configuration and coordination issues.
- **Standards:** They describe mandatory essential requirements that specify the rules for interoperability, and the compliance with operational and technical requirements.
- **Procedures:** They are concerned with the implementation rules needed to operate and exploit a system.

The *architecture of a system* is the structure or structures of the system that comprise the modules, the externally visible properties of those modules, and the relationships among them. Using metrics to assess the behaviour of the key quality attributes of these modules (and the relationships between them) is a daunting task. An architectural perspective helps to organise the complexity of the interoperability challenge in ways that can lead to more coherent treatments.

There are a number of architectural characteristics that can be used as a basis for reasoning about what might be considered appropriate quality attributes that can be measured. These include interfaces and layers, standards, data interoperability and architectural styles. (See Appendix A for more information.)

Data interoperability addresses the issues that arise when the designers of individual systems will make locally optimal decisions about data definitions and formats. Data formats resulting from such local decisions may not be compatible when operational requirements dictate that a network of systems is called upon to interoperate. Thus architectural design must provide guidance to developers to minimize the applications-layer incompatibilities that inevitably arise when systems with different purposes must communicate with each other. In OATA, the logical architecture provides a single data definition for the interfaces required for interoperability and uses object orientation to encapsulate internal data structure.

An architecture style is a family of architectures constrained by component/connector vocabulary, topology, semantic constraints and analysis methods supported [Shaw 95]. The styles are similar to design patterns but cover larger chunks of architecture, capturing a recurring solution to a recurring problem. Choosing an architectural style for a system is usually the earliest of the early design decisions and can be used to address interoperability.

2.3 Conducting an Inspection of an Architecture

2.3.1 Introduction

One of the most cost effective ways of assessing the quality attributes² of a non-executable architecture is to carry out a formal inspection of the architecture to identify and recognise the information associated to those quality attributes.

There are two main issues related to an architecture inspection: the identification of an “optimal critical mass” of information, and the generation of dependable and useful results. Although there are several techniques that can address these issues, this document focuses only on combined use of Inspection Guidelines and Inspection Reviews.

Experts that are knowledgeable either in OATA or on the Architectural Styles, Tactics and patterns must carry out the architecture inspection process. This process will never be a simple “follow the guidelines” process since it will rely strongly on the experience and knowledge of its participants. Nevertheless, the process that is presented in this document, tries to minimise this effect through the use of inspection guidelines.

The following paragraphs summarise the elaboration of Inspection Guidelines, and the scope of an Inspection Review to present recommendations about how to apply the Inspection Guidelines.

2.3.2 Elaboration of Inspection Guidelines

The Inspection Guidelines will provide guidelines on how to find and recognise the information in OATA logical architecture. The guidelines consist of:

Introduction:

Describes the interest in the entity (what entity do we mean here – part of the OATA architecture or quality characteristic being inspected?) and explains the relevant factors that support this interest.

Instructions:

Describes which parts of OATA logical architecture the inspection team will inspect, how to read those parts and how to identify and extract the appropriate information.

Set of questions framed in a procedural manner:

To support the assessment made by the inspection team while identifying and extracting the information.

The Inspection Guidelines will be produced by the Validation Team (composed of OATA Architecture experts and Validation experts). This same team should also decide on the logical entities that will be inspected.

2.4 Inspection Review

The Inspection review is relatively straightforward. It consists of four steps:

1. Organise and plan the inspection.

Use the Inspection Guidelines to review the architecture.

Identify risks and potential problems.

Document the results of the inspection process.

² In this context the Quality Attributes are assumed to be the implementation hypotheses that are embedded in the OATA architecture

3 PROCESSES

3.1 Introduction

The methodology consists of three serial processes, which have to be executed in sequence because the output of one is used as input to the next.

The first process, “Preparation”, is responsible for the planning of a validation cycle and ensures that the Validation Team is well prepared and each team member is given assigned responsibilities. The scope and purpose of the current validation must be identified with the OATA architecture version to be validated. The results and other information about the validation cycle shall be documented and stored in a repository. Therefore a repository must be identified or created in the beginning of a validation cycle.

The second process, “Execution”, handles the actual process of identifying the Architectural Tactics, Interoperability patterns and System interoperability included in the OATA logical architecture. These must be well documented and stored in the repository.

And finally, the last process to be executed, “Analysis and Reporting”, the found tactics and patterns are assessed and the Validation Team shall propose necessary recommendations how OATA logical architecture could be improved. The validation result shall be published in one or several Validation Reports that can be distributed to the Stakeholders and the results can be used to refine the OATA architecture.

An overview of the processes and their activities are shown in Figure 3.

The required information for each process is described in the process diagrams, and these diagrams also indicate what information is created or refined by the processes. An overview of the information flow is shown in Figure 4.

The processes are controlled and supported by different roles, information and physical resources. An overview of these resources is shown in Figure 5.

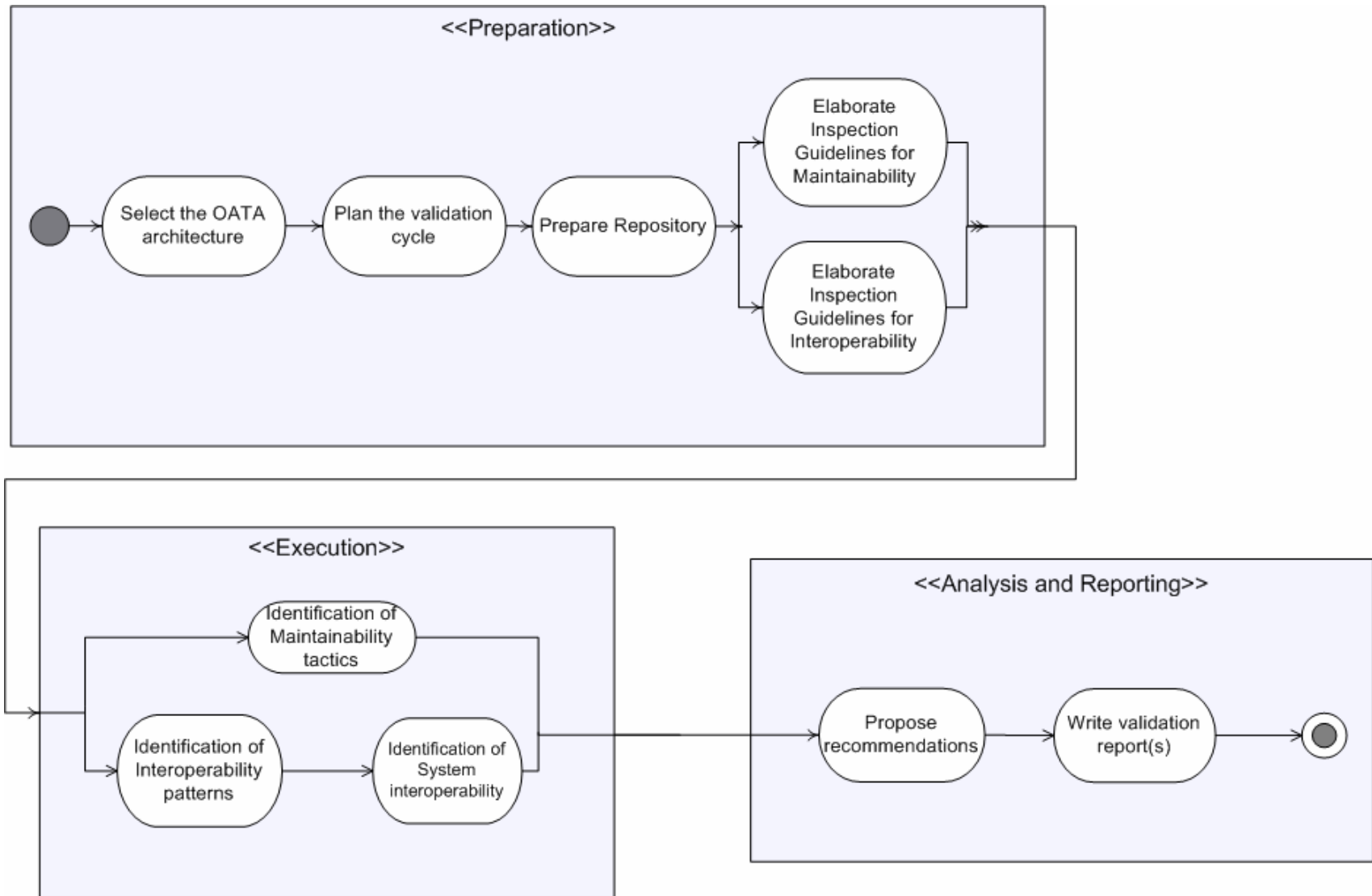


Figure 3: Process Overview

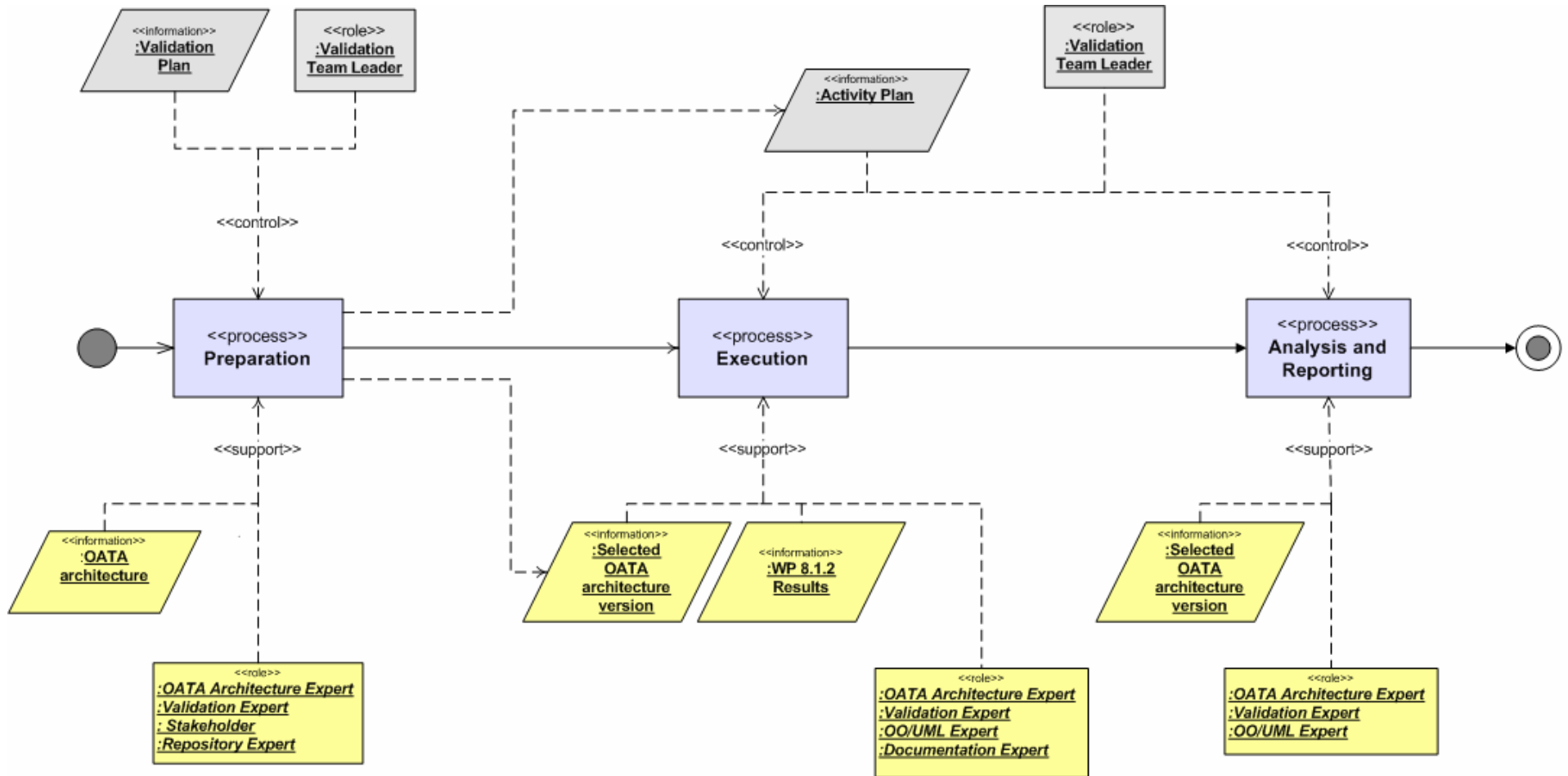


Figure 4: Resource Overview

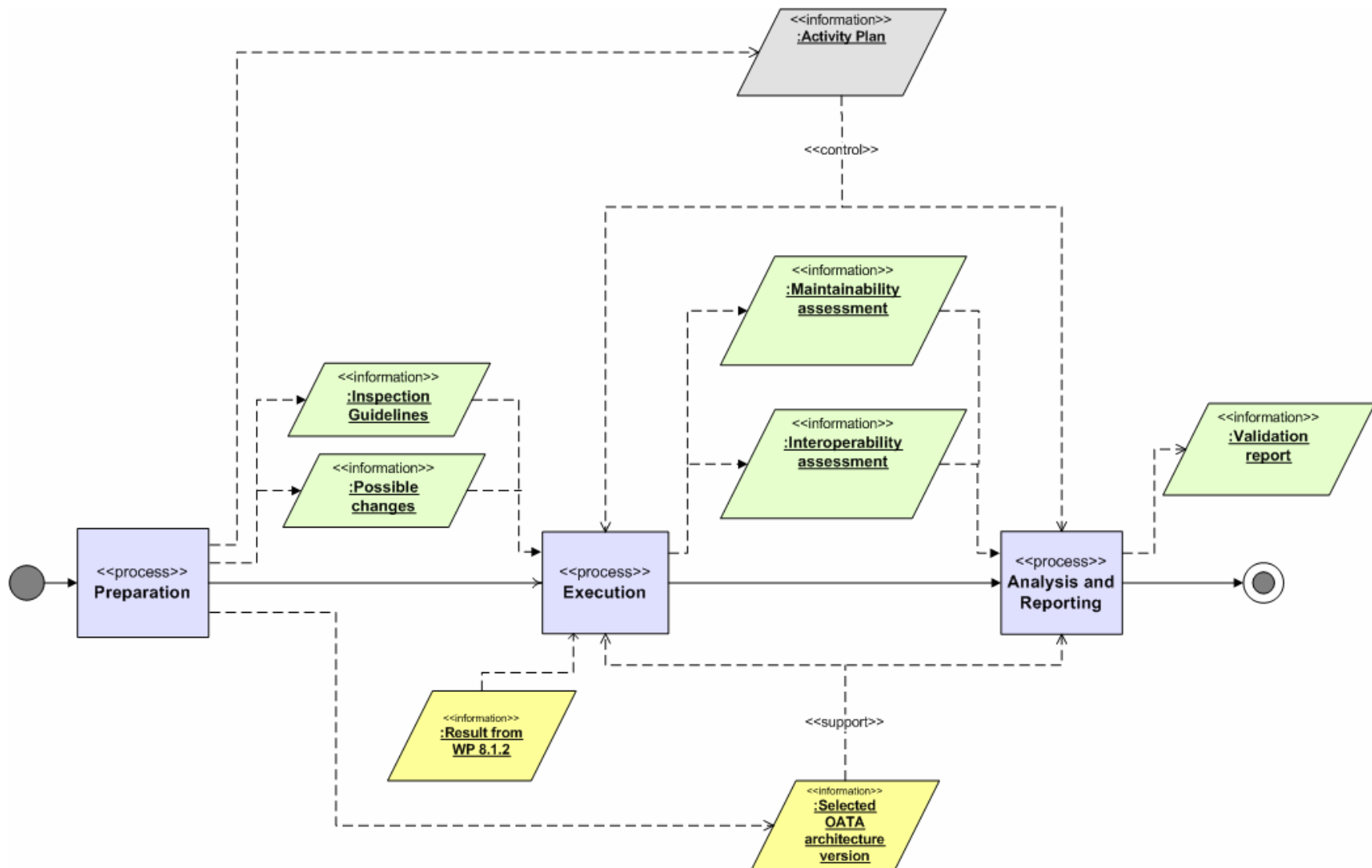


Figure 5: Information Flow Overview

3.1.1 Required Roles

The table shows the different roles and their responsibility and requirements for the validation methodology. This set of roles shall be included in the Validation Team. Each role will be assigned to one of several resources and these resources are referred to as the Validation Team Members.

Validation Team		
Role	Responsibilities	Requirements
Validation Team Leader	<p>Is responsible for the control of the validation process and team management.</p> <p>Is responsible for the Activity Plan.</p> <p>Is responsible for the Validation Reports.</p> <p>Is responsible for the Repository and the guidelines</p>	<p>Good knowledge of the OATA project and organization. Good knowledge of the validation methodology described in this document.</p>
Validation Expert	<p>Is responsible for the Inspection Guidelines.</p> <p>Is responsible for the performance of workshops</p>	<p>Expert knowledge about validation and the OATA Validation Methodologies.</p>
OO/UML Expert	<p>Shall assist the Validation Expert.</p>	<p>Expert knowledge about UML, object-orientation and patterns.</p>
OATA Architecture Expert	<p>Is responsible for the OATA architecture</p>	<p>Expert knowledge about the OATA architecture. Full access to the OATA architecture. Mandate to select version to validate</p>
Stakeholder	<p>Identify operational changes that can happen e.g. 2011 but have not been described in the operational concept that was used during the development of the OATA Architecture.</p>	<p>Expert knowledge about the operational concepts and part of the discussion for the future.</p>
Documentation Expert	<p>Is responsible for the documentation of the workshops.</p>	<p>Expert knowledge about documentation and knowledge about the methodology and the repository.</p>
Repository Expert	<p>Is responsible for the Repository</p>	<p>Expert knowledge and access to the OATA repository.</p>

3.2 Preparation

3.2.1 Introduction

The objective of the “Preparation” process is to prepare the necessary information required in the “Execution” and the “Analysis and Reporting” processes.

The Preparation process consists of the following activities:

- Select the OATA architecture version
A released version of the OATA architecture shall be selected for validation during a validation cycle. All parts of the OATA logical architecture that, for some reason, shall not be included in the assessment, shall be decided and documented.
- Plan the validation cycle
The Validation Team Leader plans the validation cycle within the framework of the OATA overall Validation Plan (input information) and makes sure that the validation team members are available and assigns responsibilities. A repository shall be selected or created.
- Prepare the repository
A suitable repository to store the validation results shall be created (or identified) and prepared.
- Elaborate Inspection Guidelines for Maintainability
Decide inspection tasks and elaborate guidelines for the inspection of the OATA Architecture regarding Maintainability.
- Elaborate Inspection Guidelines for Interoperability
Decide inspection tasks and elaborate guidelines for the inspection of the OATA Architecture regarding Interoperability.

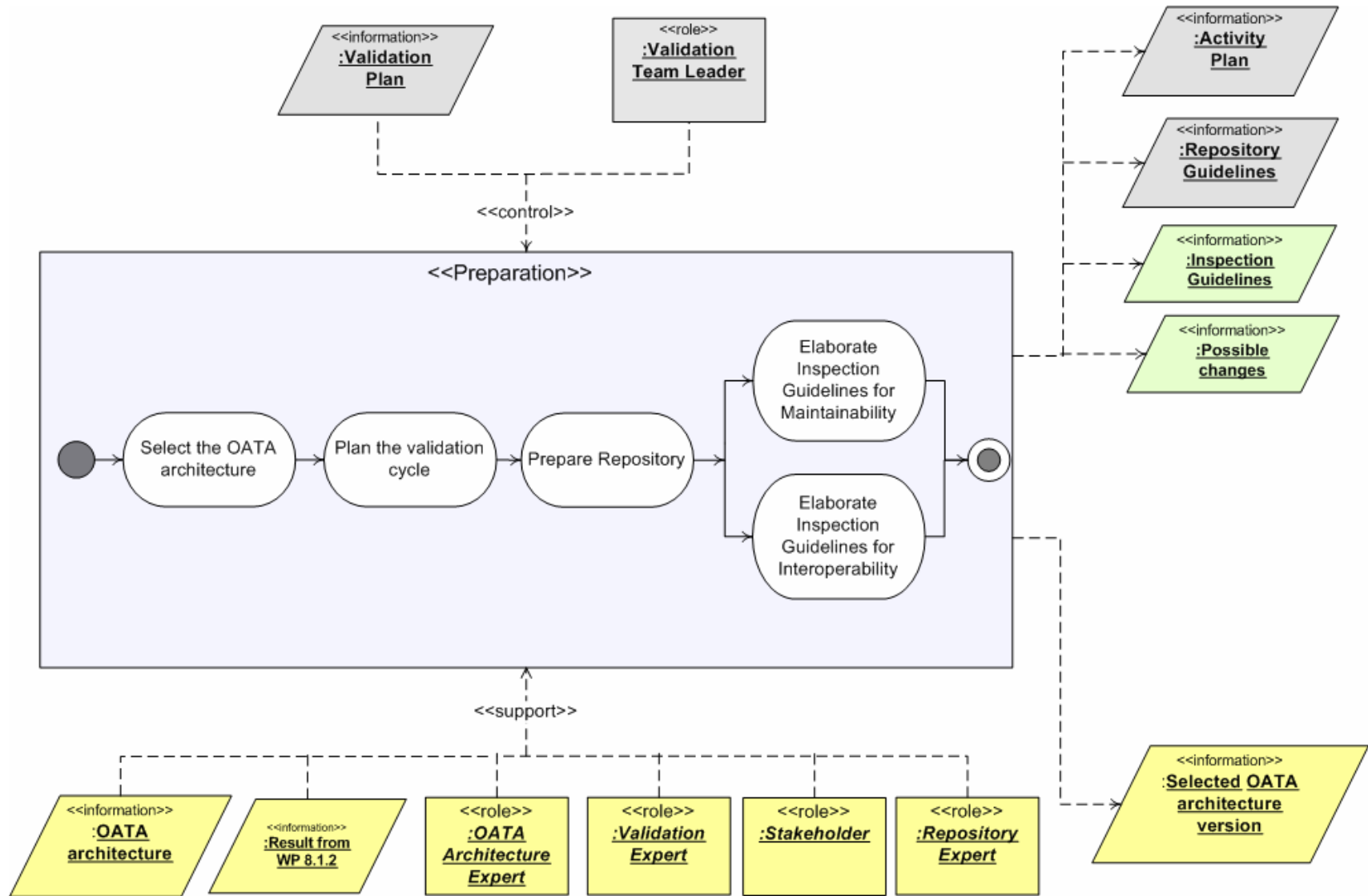


Figure 6: Preparation

3.2.2 Required Resources

The table shows the planned participant resources for the activities in “Preparation”.

The highlighted markings are the role responsible for the activity.

Activities	Roles				
	Validation Team Leader	Validation Expert	OATA Architecture Expert	OO/UML Expert	Repository Expert
Select the OATA architecture version			X		
Plan the validation cycle	X	X			
Prepare repository		X			X
Elaborate Inspection Guidelines for Maintainability		X	X	X	X
Elaborate Inspection Guidelines for Interoperability		X	X	X	X

3.2.3 Activity: Select the OATA Architecture

3.2.3.1 Objective

The objective of this activity is to select the OATA architecture version and subset for the current validation cycle.

3.2.3.2 Rationale and Context

The development of the OATA architecture is an iterative work process with regular version releases. One specific version of the OATA architecture must be selected in order to ensure that it is exactly the same version that is during the validation cycle. This is also necessary for the traceability to the validated version when future analysis and comparisons are done. A subset of the OATA Architecture can also be selected when a validation will not be performed of the whole model.

3.2.3.3 Description

The OATA Architecture Expert shall select and document the OATA architecture version that shall be validated. The selected version must be clearly identified and accessible for the Validation Team. When the OATA architecture shall be validated partly, the OATA Architecture Expert shall select and document a relevant subset of the OATA architecture. The OATA Architecture Expert shall also document the reason why certain parts have been selected for the current validation cycle.

3.2.3.4 Examples and Recommendations

The selected architecture version will refer to a specific OATA baseline, for example: OATA Architecture iteration 7.

The subset can be either all clusters or a set of clusters in the OATA Architecture.

3.2.3.5 Product

The result of this activity is a selected version of the OATA architecture for the current validation cycle and the subset to validate.

3.2.3.6 Resources

The required resources for this activity are shown in the table below.

The highlighted markings are the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
OATA Architecture Expert	X	X	OATA architecture	The selected OATA architecture version The selected OATA architecture subset

3.2.4 Activity: Plan the Validation Cycle

3.2.4.1 Objective

The objective of this activity is to ensure that required resources are available and a properly trained validation team for the assignment.

3.2.4.2 Rationale and Context

Before the validation can be initiated it is important that the Validation Team Leader ensures that the validation team members are available and well prepared for the validation. To have effective use of the validation team members, the Validation Team Leader should prepare a relatively detailed activity plan.

3.2.4.3 Description

The Validation Team leader is responsible for the planning of the validation cycle and that each team member is well prepared.

The Validation Team Leader shall perform the following tasks:

1. Plan the activities in the validation process and document the planning in an Activity Plan.
2. Introduce the validation methodology to the team members. Each validation team member shall understand the processes and their activities.
3. Assign roles and responsibilities to each validation team member.

3.2.4.4 Examples and Recommendations

Detailed and early planning of the validation cycle is important for the final validation results. The Activity Plan should contain at least the following information:

- List of reference material
- Description of the activities
- Description of the Workshops
- Resource description

- Resource plan
- Schedule

3.2.4.5 Product

The expected result of this activity is:

- Well prepared validation team members with assigned responsibilities.
- An activity plan for the current validation cycle.

3.2.4.6 Resources

The required resources for this activity are shown in the table below.

The highlighted markings are the role responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Team Leader	X	X	Validation Plan	Activity Plan
Validation Expert		X		

3.2.5 Activity: Prepare the repository

3.2.5.1 Objective

The objective is to prepare suitable information storage. All information gathered during the validation shall be stored in the repository.

3.2.5.2 Rationale and Context

A repository shall be selected or created to be used for documentation of the validation cycle. Guidelines for its usages should also be prepared and made available for the validation team members.

3.2.5.3 Description

The validation repository can be either a set of Word or Excel templates or it can be a database. It is important that the repository is coordinated with the other validation methodologies in order establish common search criteria and storage layout.

The **first task** is to identify existing repositories from other validation cycles (e.g. Pilot Validation). If none exists the repository has to be created. Identify suitable type of repository and collect type of information that shall be stored in the repository by examining the results from the validation.

The **second task** is to create (or identify) repository guidelines to how it shall be used.

3.2.5.4 Examples and Recommendations

A repository should be a well-structured place to store the documentation of identified architectural tactics and other important information produced during a validation cycle. It can be for example an Excel-sheet, database or a folder in a file system (or a combination).

Following criteria should be considered:

- The results must be easy to identify.

- Common information like Validation Aim shall be stored in the same way as Validation Aims from other validation aspects, e.g. functional and non-functional Validation Aims.
- The documentation of identified architectural tactics should be stored in the repository in order to facilitate analysis.

If a repository is already available (for example from previous validation iterations) for storing validation data, then this should be used. It is essential that the validation data results are fully identifiable and traceable.

The possibility to search information about validation results and different validation aspects (like Validation Aims, Conclusions, Recommendations from Functional, Non Functional and Tactics) is more important than how effective the validation repository is to use during a validation cycle.

3.2.5.5 Product

The expected result from this activity is:

- A prepared repository for storage of the validation data results
- Repository guidelines

3.2.5.6 Resources

The required resources for this activity are shown in the table below.

The highlighted markings are the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information	Physical
Repository Expert	X	X		Repository guidelines	Repository
Validation Expert		X			

3.2.6 Activity: Elaborate Inspection Guidelines for Maintainability

3.2.6.1 Objective

The objective of this activity is to elaborate Inspection Guidelines for the identification of maintainability tactics used in the OATA Architecture and to detect possible maintainability problem areas.

3.2.6.2 Rationale and Context

The tactics for affecting maintainability are organised into three categories which are those that:

- Localizes possible modifications.
- Restricts the visibility of responsibilities.
- Prevents the ripple effects.

Each of these categories is described in the Appendix A.

3.2.6.3 Description

The identification of tactics is performed through an inspection of architecture components. This inspection should be geared towards the identification of the tactics presented in the table below. The inspection team will accomplish this by identifying the architectural components that meet the tactics requirements.

The **first task** is to plan the inspection and elaborate Inspection Guidelines (see chapter “Conducting an inspection of architecture”) regarding each selected inspection task (see tables below). These tactics are further explained in Appendix A. Use the tables below for the selection of inspection tasks and the recommended approach. The Inspection Guidelines should contain at least:

- Description of each inspection task
- Definition of the validation aim for each inspection task
- Key questions to assist the validation
- Prepared tables to assist the documentation and to assist the validation performance. The prepared tables should be inline with the repository.

The **second task** is to collect basic data from the OATA Architecture and prepare the result tables to assist and make the execution activities of the validation efficient.

In the table below are the inspections tasks for tactics localizing expected modifications described. Please note that not all of the inspection tasks are recommended for the inspection of the OATA architecture.

Tactics for localizing expected modifications			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Maintain semantic coherence	Yes	Identify all responsibilities for each module and assess the semantic coherence.	
Isolate the possible changes	Yes	Involve suitable Stakeholder to identify operational (or technical) changes that will affect the OATA Architecture. The document Operational Improvements in the Eurocontrol ATM 2000+ Strategy can also be of use. Identify modules that will be affected by the identified changes and assess the identified modules' stability.	See Examples and Recommendations
Raise the abstraction level	No	-	
Limit options	No	-	
Abstract common services	Yes	Identify similar services that are provided in two or more modules.	

In the table below are the inspections tasks for identifying tactics restricting the visibility of responsibilities described. Please note that none of the inspection tasks are recommended for the inspection of the OATA architecture.

Tactics for restricting the visibility of responsibilities			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Hide information	No	-	See Appendix A
Maintain existing interfaces	No	-	See Appendix A
Separate the interface from the implementation	No	-	See Appendix A

In the table below are the inspections tasks for tactics preventing the ripple effect described. Please note that not all of the inspection tasks are recommended for the inspection of the OATA architecture.

Tactics for preventing the ripple effect			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Break the dependency chain	Yes	Identify the existence and appropriate use of: <ul style="list-style-type: none"> • Publish-subscribe pattern • Repository and identify the absence of: <ul style="list-style-type: none"> • Platform Specific Models (PSM) elements The metric CYC from Architecture Structure Assessment, [OATA WP 8.1.2], detects where dependency chains have not been broken and tactics should be applied.	
Make the data self-identifying	No	-	See Appendix A
Limit communication paths	Yes	Identify coupling between modules and build a communication path.	

In the table below are the inspections tasks for validating the modelling guide regarding maintainability described.

Validating the modelling guide regarding Maintainability			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Validate the Modelling guide	Yes	Inspect the modelling guide and identify tactics to enhance the maintainability of the OATA Architecture. <ul style="list-style-type: none"> - Maintain semantic coherence - Isolate the possible changes - Abstract common services - Break the dependency chain - Limit communication paths 	

3.2.6.4 Examples and Recommendations

Results from Architecture Structure Assessment, [OATA WP 8.1.2], are recommended to be used to identify problem areas and to build the dependency chains, especially for the inspection tasks “Limit communication paths” and “Break the dependency chain”. The Inspection Guidelines shall include identification of useful results from the metric measurement during Architecture Structure Assessment. The validation results and assessments should be correlated with validation results from the Architecture Structure Assessment.

The inspection task “Isolate the possible changes” requires input from the Stakeholders. The Stakeholders shall identify operational or technical changes that are not foreseen during the development of the OATA Architecture and which may affect the OATA Architecture. The effect of such operational or technical change should be minimized as far as possible.

Examples of possible operational or technical changes are:

- *“ASAS may not be implemented to 2011”*
- *“Take-off sequence may also consider the aircraft leaving the TMA”.*
- *“The Arrival management and Departure Management maybe integrated”*
- *“Capacity planning may also include capacity figures for the Taxiways”*

While elaborating the Inspection Guidelines, existing guidelines should be identified and used as templates (e.g. from the Pilot Validation or other cycles).

3.2.6.5 Product

The result of this activity is the Inspection Guidelines for Maintainability that shall be used for inspecting the OATA Architecture.

3.2.6.6 Resources

The required resources for this activity are shown in the table below.

The highlighted marking is the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	The selected OATA architecture version and subset	Inspection Guidelines for Maintainability
OO/UML Expert		X		
OATA Architecture Expert		X		List of possible changes
Stakeholder		X		

3.2.7 Activity: Elaborate Inspection Guidelines for Interoperability

During the Pilot Validation it was discovered that validation of interoperability as it is described in this document is not effectively applicable on the OATA Architecture. The methodology of interoperability validation is under re-construction and will be changed.

3.2.7.1 Objective

The objective of this activity is to elaborate Inspection Guidelines for the identification of interoperability patterns used in the OATA Architecture and to detect possible interoperability problem areas.

3.2.7.2 Rationale and Context

The architectural elements supporting interoperability must be identified for the assessment of interoperability of the OATA logical architecture.

One of the main difficulties of assessing interoperability from an architectural point of view is that even though architectural decisions influence greatly the interoperability of systems, interoperability is often only visible when the system is designed.

Three patterns have been identified (through research and implementation experience) as having a direct relationship with the interoperability of a system:

- Translator patterns
- Controller patterns
- Extender patterns

The existence of these patterns does not necessarily imply that an architecture is fully interoperable, but their absence indicates potential problems.

Appendix A contains their definition and some ideas regarding their usage which may be limited in the OATA context, which addresses only the application part of a platform independent specification model.

3.2.7.3 Description

The identification of interoperability patterns is performed through an inspection of the architecture components. This inspection should be geared towards the identification of the three stated patterns. The inspection team should identify the architectural components that meet the patterns requirements.

The **task** is to plan the inspection and elaborate Inspection Guidelines (see chapter “Conducting an inspection of an architecture”) regarding each selected inspection task (see tables below). These patterns are further explained in Appendix A.

Patterns			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Translator Patterns	No	-	
Controller Patterns	No	-	
Extender Patterns	No	-	

3.2.7.4 Examples and Recommendations

Even though this activity is strongly linked with the next activity (*Identification of System Interoperability*) it is a separate activity in this methodology to maintain the description similar to the previous activities.

The reader should note that not all possible interoperability patterns (e.g. Semantic Interoperability) have been included in this text. This is because OATA does not have the necessary information (namely the existence of an ontology) to perform this assessment, and because OATA has chosen a clean sheet approach (taking into account more or less informally the way things are done today). Nevertheless, Appendix A presents a brief discussion about semantic interoperability and how to assess it.

To avoid “overdoing” an inspection of the OATA logical architecture, the validation team should find a minimal set of patterns. The patterns that are described in this section are fairly common, and it could be possible to identify some of them that would not be directly applicable to the interoperability. This risk is minimised through the use of the Inspection Guidelines (as mentioned in Appendix A).

3.2.7.5 Product

The result of this activity is the Inspection Guidelines for Interoperability that shall be used for inspecting the OATA Architecture.

3.2.7.6 Resources

The required resources for this activity are shown in the table below.

The highlighted marking is the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	The selected OATA architecture version	Inspection Guidelines for Interoperability
OO/UML Expert		X		
OATA Architecture Expert		X		

3.3 Execution

3.3.1 Introduction

The objective of the Execution process is to perform an inspection of the OATA architecture.

The Execution process consists of the following activities:

- *Identification Maintainability Tactics*

Inspection of the OATA logical architecture in order to identify tactics regarding maintainability made during development of the architecture.

- *Identification of Interoperability Patterns*

Inspection of the OATA logical architecture in order to identify a minimal set of architectural elements related to interoperability.

- *Identification of System Interoperability*

Inspection of the OATA logical architecture in order to identify tactics regarding solved interoperability conflicts during the development of the architecture.

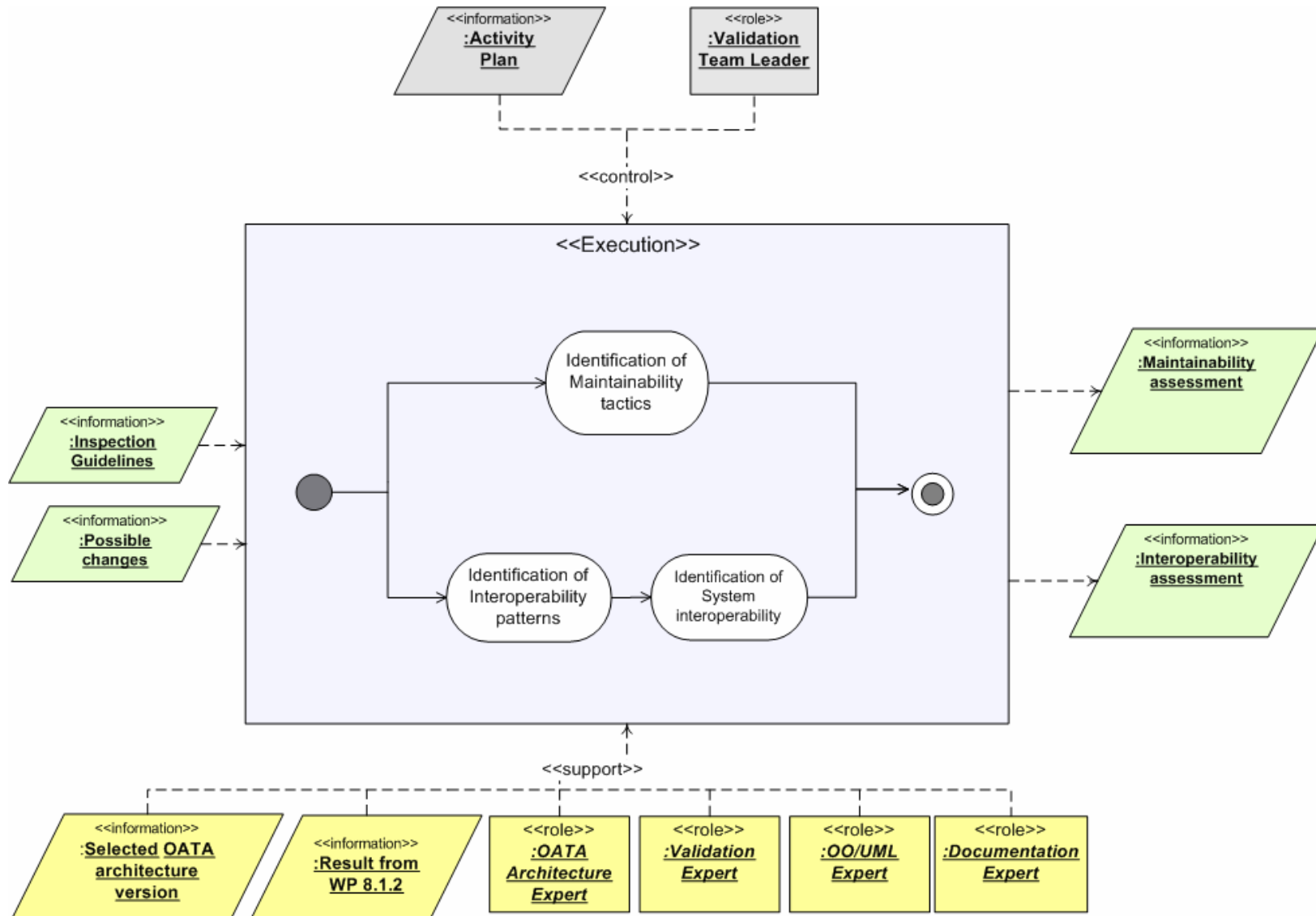


Figure 7: Execution

3.3.2 Required Resources

The table shows the planned participating resources for the activity in “Execution”.

The highlighted markings are the role main responsible for the activity.

Activities	Roles				
	Validation Team Leader	OATA Architecture Expert	Validation Expert	OO/UML Expert	Documentation Expert
Identification of Maintainability Tactics		X	X	X	X
Identification of Interoperability Patterns		X	X	X	X
Identification of System Interoperability		X	X	X	X

3.3.3 Activity: Identification of Maintainability Tactics

3.3.3.1 Objective

The objective of this activity is to use the Inspection Guidelines for Maintainability to identify maintainability tactics used in the OATA Architecture and to detect possible maintainability problem areas

3.3.3.2 Rationale and Context

See Rationale and Context for the activity “Elaborate Inspection Guidelines for Maintainability” section 3.2.6.2.

3.3.3.3 Description

The **task** is to use the Inspection Guidelines to identify and assess according to the inspection tasks. All risks and potential problems shall also be identified and documented.

3.3.3.4 Examples and Recommendations

Results from Architecture Structure Assessment, [OATA WP 8.1.2], are recommended to be used to identify problem areas and to build the dependency chains. Results generated in this activity should be correlated with results generated in validation of Architecture Structure Assessment.

This activity is recommended to be performed in a workshop together with one or several OATA Architecture Experts. The documentation during the workshop is very important but also difficult to perform with high quality. The result tables in the Inspection Guidelines should be of that quality that the documentation is facilitated. The analysis of the validation is facilitated if the documented results are tagged with date and time and thereby easier to recap the workshop.

It is recommended to assign the documentation to a specific role during the workshop. The documentation must however be described at the workshop to ensure consensus regarding the validation results.

3.3.3.5 Product

The result from this activity is a list and description of assessed potential architectural maintainability risk areas.

3.3.3.6 Resources

The required resources for this activity are shown in the table below.

The highlighted markings are the role responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	The selected OATA architecture version	Maintainability assessments
			Inspection Guidelines for Maintainability	
			List of possible changes	
OO/UML Expert		X	Result from Architecture Structure assessment should be used and correlated with this activity	
OATA Architecture Expert		X		
Documentation Expert		X		

3.3.4 Activity: Identification of Interoperability Patterns

During the Pilot Validation it was discovered that validation of interoperability as it is described in this document is not effectively applicable on the OATA Architecture. The methodology of interoperability validation is under re-construction and will be changed.

3.3.4.1 Objective

The objective of this activity is use the Inspection Guidelines for Interoperability to identify the occurrence of a minimal set of architectural elements that are related to interoperability, with the potential for forecasting the interoperability of an implementation of the OATA logical architecture.

3.3.4.2 Rationale and Context

See Rationale and Context for the activity “Elaborate Inspection Guidelines for Interoperability” section 3.3.4.2.

3.3.4.3 Description

The **task** is to use the Inspection Guidelines to identify and assess according to the inspection tasks. All risks and potential problems shall be identified and documented.

3.3.4.4 Examples and Recommendations

No information.

3.3.4.5 Product

The result from this activity is a list of identified and assessed architectural constructs regarding interoperability patterns.

3.3.4.6 Resources

The required resources for this activity are shown in the table below.

The highlighted markings are the role responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	The selected OATA architecture version	Interoperability assessments
OO/UML Expert		X		
OATA Architecture Expert		X		
Documentation Expert		X		

3.3.5 Activity: Identification of System Interoperability

During the Pilot Validation it was discovered that validation of interoperability as it is described in this document is not effectively applicable on the OATA Architecture. The methodology of interoperability validation is under re-construction and will be changed.

3.3.5.1 Objective

The objective of this activity is to use the Inspection Guidelines for Interoperability and inspect if the OATA logical architecture is interoperable from an architecture system point of view.

3.3.5.2 Rationale and Context

See Rationale and Context for the activity “Elaborate Inspection Guidelines for Interoperability” section 3.3.4.2.

3.3.5.3 Description

Different types of interactions occur between architecture modules. These interactions occur when their expectations affect each other. Even though there can be module-to-module interactions, middleware-to-requirements interaction, and so on, the focus of this activity is on the “requirements-to-module” interactions. Only when an interaction results in interoperability conflict it becomes paramount to interoperability analysis.

Using available experience in the development and integration of systems, it is possible to identify several types of “typical” interoperability conflicts that arise when two systems are integrated. These conflicts are addressed best through the application of patterns and strategies during the elaboration of the architecture. The existence of the patterns indicates that the issues have been addressed, and thus that the system has a higher chance of being interoperable.

The result from the activity “Identification of Interoperability Patterns” shall be used in this activity to check whether each one of the following interoperability conflicts is addressed. Seven broad categories of interoperability conflict resolution are described below. Each

category represents a piece of the overall solution (an integration sub-architecture) used to resolve interoperability conflicts [Davis 00]:

- **Variance** – all interoperability conflicts that necessitate hiding the implementation of the components involved fall under this category. A translator such as a bridge or adapter would be implemented to resolve a transparency problem.
- **Coordination** – all interoperability conflicts that require only a decision-making strategy, be it directing data or choosing the next component to execute, comprise this category. The implementation of a controller such as a façade or a selector resolves the determination problem.
- **Deficiency** – all interoperability conflicts that have need of some device to perform tasks not covered by translation or control fall in this category. Extension, such as security checking, use of a buffer or repository, provides the additional functionality to resolve these conflicts.
- **Coordination with Variance** – all interoperability conflicts that are in need of interface transparency and decisive routing are encompassed by this category. The elements of translation and control are combined to fix arbitration conflicts, as in a mediator [BMRSS96] or an application gateway.
- **Coordination with Deficiency** – all interoperability conflicts needing determination or direction plus some means or device to store and exchange information. A shared repository using a controller to direct the information and an extender to store it may resolve this type of conflict.
- **Variance with Deficiency** – all interoperability conflicts having both transparent implementation needs and requiring additional services for seamless integration are contained in this category. Translation joined with extension provides for this functionality, e.g. a proxy.
- **Coordination with Variance and Deficiency** – all interoperability conflicts that demand arbitration plus must use resources to allow communication to flow without problems are harboured in this category. To gain the full functionality, a combined translation, control and extension mechanism must be created as a manager. This can be found in the Broker pattern that underlies middleware such as CORBA.

The validation team will perform an inspection of the architecture to identify how and where the architecture addresses each one of the potential interoperability conflicts, as far as it is possible to do so at the logical level.

The table below includes the different inspections tasks for identifying interoperability conflicts.

Interoperability Conflicts			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Variance	No	-	
Coordination	No	-	
Deficiency	No	-	
Coordination with Variance	No	-	
Coordination with Deficiency	No	-	
Variance with Deficiency	No	-	

Interoperability Conflicts			
Inspection Tasks	Recommended to OATA	Approach	Remarks
Coordination with Variance and Deficiency	No	-	

3.3.5.4 Examples and Recommendations

To minimise the number of patterns (Translator, Controller and Extender) to be found, the validation team should find modules that are likely to be part of an interoperability conflict. Typically these conflicts can be found on the boundary of the system, mainly on the interfaces with the actors, external systems and sensors. Since OATA is defined as a system of systems, special attention should be given, where the validation team expects, that collaborating instantiations of the modules will be implemented on different geographical locations.

3.3.5.5 Product

List and description of potential system interoperability risk areas.

3.3.5.6 Resources

The required resources for this activity are shown in the table below.

The highlighted marking is the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	Patterns identified in the activity "Identify Interoperability Patterns"	Interoperability assessments
OO/UML Expert		X		
OATA Architecture Expert		X		
Documentation Expert		X		

3.4 Analysis and Reporting

3.4.1 Introduction

This chapter describes the activities related to the analysis and reporting of the results obtained during the inspection of the selected parts of OATA logical architecture. The analysis will be focused on the identification of the impacts of the identified (and of the not identified) tactics and patterns.

The process consists of the following activities (see Figure 8 below):

- **Propose Recommendations**

The result from Execution process is analysed and the validation team proposes recommendations how OATA logical architecture could be improved.

- **Write the Validation Reports**

The results and conclusions of the validation cycle are documented in Validation Report(s).

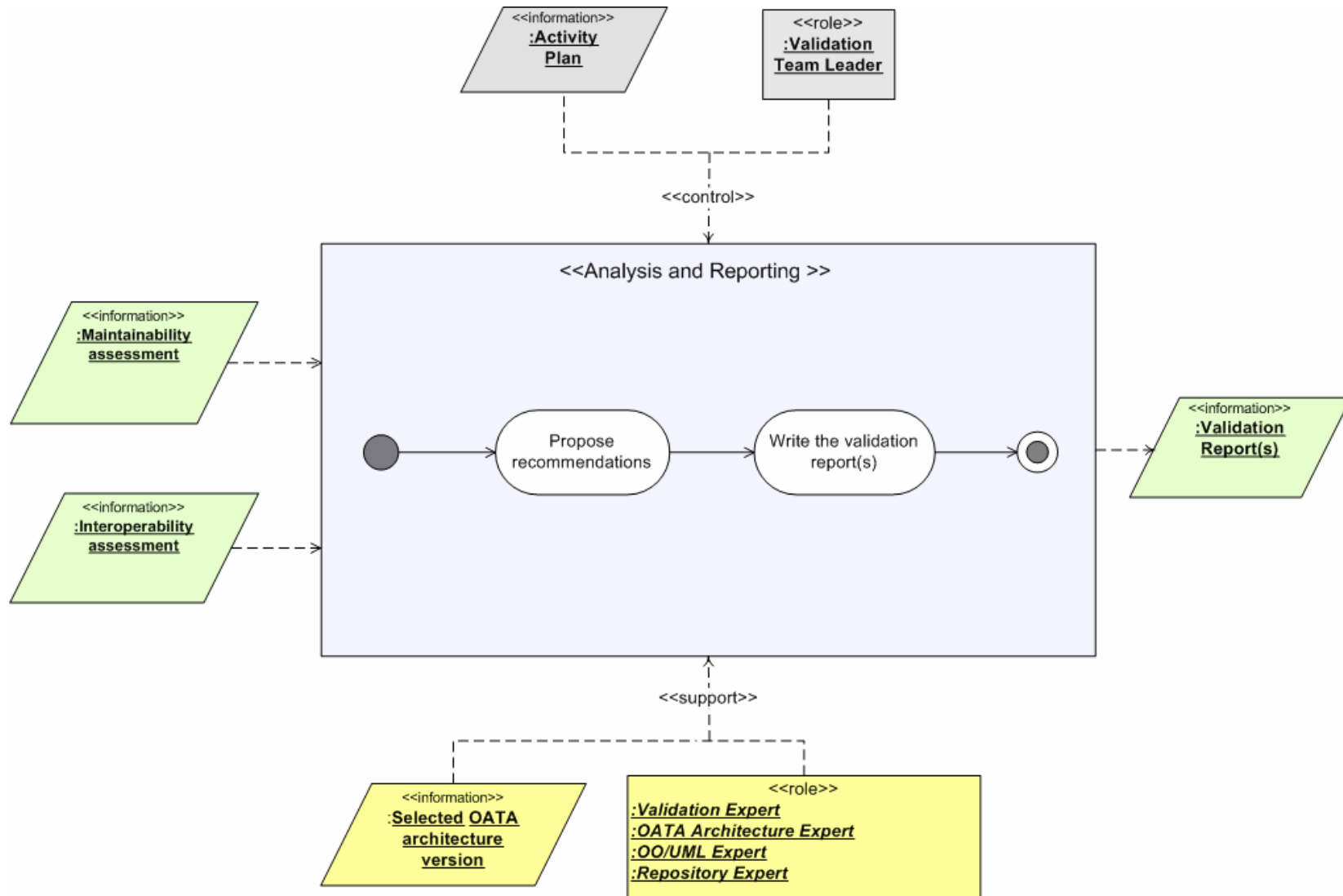


Figure 8: Analysis and Reporting

3.4.2 Required Resources

The table shows the planned participating resources for the activity in “Analysis and Reporting”.

The highlighted markings are the role main responsible for the activity.

Activities	Roles			
	Validation Team Leader	OATA Architecture Expert	Validation Expert	OO/UML Expert
Propose Recommendations	X	X	X	X
Write the validation report(s)	X	X	X	X

3.4.3 Activity: Propose Recommendations

3.4.3.1 Objective

The objective of this activity is to analyse and assess the information obtained during the inspections and to propose recommendations regarding the improvement of the OATA logical architecture.

3.4.3.2 Rationale and Context

The identification of architectural tactics is extremely difficult due to the lack of an implementation model, and also to the potential existence of non-documented implicit hypotheses. To overcome this problem, the guidelines propose a method based on the identification of the main architectural styles, tactics and patterns that are associated to the three areas that have a greater impact on the implementation: maintainability, efficiency and interoperability. The identification of these elements is not enough; the analysis must include also the recommendations and the assessment (based on expert judgment) of their importance. Thus, through this activity the validation team has an opportunity to collect and review the different information and to produce an assessment of its importance.

3.4.3.3 Description

The **main task** of this activity consists of making comments on each of the results obtained during the inspection of the selected subsets OATA logical architecture. It is very important to understand that this activity is dependent in the experience of the analyst, as it involves the interpretation of the results.

All of the comments shall be collected, compiled and summarized for a better understanding and to be used as reference to make further decisions about the improvements of the architectural aspects of the system.

The defined validation aims for each inspection task shall be used for conclusions.

3.4.3.4 Examples and Recommendations

No information.

3.4.3.5 Product

The output of this activity is recommendations how OATA logical architecture could be improved.

3.4.3.6 Resources

The required resources for this activity are shown in the table below.

The highlighted marking is the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Team Leader	X	X	Maintainability assessments Interoperability assessments	Recommendations
Validation Expert		X		
OO/UML Expert		X		
OATA Architecture Expert		X		

3.4.4 Activity: Write the Validation Report(s)

3.4.4.1 Objective

The collected information and recommendations from the performed OATA validation cycle needs to be summarized and prepared for presentation to the OATA Stakeholders.

3.4.4.2 Rationale and Context

The information produced during the previous steps and activities must be structured in such a way that it is easily accessible and interpreted. Documenting and distributing the validation results precipitates two effects. First, it results in a closure of the validation by relating the final results to the initial presentation. Second, it elevates the risks that were uncovered to the attention of the management. What might otherwise have seemed to a manager like a complicated technical issue is now unambiguously identified as a threat to system qualities.

3.4.4.3 Description

The validation report should be prepared during the planning phase and continuously updated to make this activity easier and more efficient. When the validation report is ready it should be sent to the validation team including Stakeholders for review. The validation report shall be updated and the reviewers should be reflected in the document history.

The following paragraphs present and describe the different areas that should be addressed by a validation report. Each one of the following items corresponds to a section of the Validation Report template used in the Pilot Validation.

Report Section	Comment	Validation Process
1. Introduction and Document Background.	Establish the information needed to understand the report. Project background, glossary, definition of terms, references, etc.	Preparation

Report Section	Comment	Validation Process
2. Summary of validation strategy and planning	Describe the overall planning and strategy of the validation	
3. Conduct of validation exercise	Describe deviations, the overall schedule and validation team	
4. Validation results	Describe the results: <ul style="list-style-type: none"> • Validation Aims • Validation Objectives • Operational Threads • Specific validation results 	Execution
5. Analysis of the validation result	Document the analysis based on the validation aims and objectives	Analysis & Reporting
6. Conclusions and Recommendations	Document conclusions and recommendations from the validation team.	
7. APPENDIX A: Detailed Validation Results and summary matrices	Synchronized with the Repository	Execution
		Analysis & Reporting

3.4.4.4 Examples and Recommendations

It is also recommended to generate a lessons-learned report, to record the main conclusions on the validation methodology itself, regardless of the concrete data obtained in the assessment process.

3.4.4.5 Product

The output from this activity and the validation cycle is the Validation Report and optionally a Lessons Learned Report.

3.4.4.6 Required Resources

The required resources for this activity are shown in the table below.

The highlighted marking shows the role main responsible for the activity.

Role	Responsible	Executor	Input Information	Output Information
Validation Team Leader	X	X	Validation results	Validation Report
Validation Expert		X		Optional: Lessons Learned Report
OATA Architecture Expert		X		
Documentation Expert		X		
Stakeholders (review)		X		

4 APPENDIX A: TACTICS AND PATTERNS

4.1 Introduction

4.1.1 Architecture Tactics

An architecture tactic is a means of satisfying a quality attribute response measure by balancing some quality aspect through architectural design decisions.

The definition has the following consequences:

- An architectural tactic bridges the architectural quality and the architectural design. It does so by specifying how quality attributes can be controlled through architectural decisions to achieve a desired response measure.
- An architectural tactic uses knowledge from various reasoning frameworks (such as performance engineering or maintainability framework) to support the creation of quality attributes that mirror the architecture being designed.

Analytic models for the various quality attributes makes it possible to identify design decisions that offer leverage for achieving quality attribute requirements. The analytic models also offer a reasoning framework for explaining how changes in the design decisions affect the quality attribute validation scenario response. Architectural tactics are part of this reasoning framework.

While tactics are motivated from an analytic perspective, they have specific realisations in the architecture. For example, while a performance engineering queuing model might only require an average execution time as input to a validation model, there are many sources of execution time that need to be derived from an architectural design. There is a many-to-one relationship between the analytic model and the corresponding architectural realisations.

Tactics are not necessarily independent. The application of a tactic may also require additional tactics to be applied. For example, the application of the “Break the dependency chain” tactic to insert an intermediary would likely require additional tactics to isolate those intermediary responsibilities.

4.1.2 Architectural Patterns

Architectural Patterns “express a fundamental structural organization schema for software systems. They provide a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.” [Buschmann 96]

4.1.3 Architecture and Interoperability

The *architecture of a system* is the structure or structures of the system that comprise the components, the externally visible properties of those components, and the relationships among them []. Using metrics to assess the behaviour of the key quality attributes of these components (and the relationships between them) is a daunting task. An architectural perspective helps to organize the complexity of the interoperability challenge in ways that can lead to more coherent treatments.

Architectures are a hierarchical description for the design of a system and in many cases describe how it will be developed, evolved, and operated. Architectures provide the underlying blueprint for the more detailed design and implementation decisions about the components of a system. When well-defined architectures exist, engineers can design individual components and builders can implement them with a high degree of confidence that the end results will work as expected and meet user needs.

There are a number of architectural characteristics that can be used as a basis for reasoning about what might be considered appropriate quality attributes that can be measured. These include interfaces and layers, standards, data interoperability and architectural styles.

When reasoning about architecture, it makes sense to strive for an information systems environment based on well-defined requirements specification, common data structures, common interface requirements, and well-specified high-level information flows. Systems constructed in accordance with such an architecture are much more likely to be adequately interoperable than those that are not. However, particularly when legacy systems are involved, these commonalities may not exist. In these cases, architectures can supply valuable guidance to isolate gaps and risks relative to interoperability.

4.1.3.1 Interfaces and Layers

The modular decomposition of systems is typically both horizontal and vertical. Vertical decomposition refers to interfaces between discrete systems within the same layer (e.g., a standard message format used by different applications to exchange information). Horizontal decomposition of functions is known as layering (e.g., the separation of bit transport technologies, transport protocol, and applications).

Interfaces

Systems that perform a variety of functions are normally composed of multiple subsystems or components. Interfaces arise whenever one subsystem or component interacts with another.

An architect that is designing and partitioning a system that is intended to interoperate with unspecified existing and future systems must consider the importance of the following:

- Interface design. Well-designed and documented interfaces that permit development programs to be divided into more manageable pieces. This results in faster development because the work of different players can proceed in parallel.
- Encapsulation. This permits modular change in version and implementation technology. By encapsulating the internal details of a system component that may change over time, interfaces allow changes in internal implementation of portions of a system to be transparent to other external systems.
- Reducing interaction. Reducing the complexity of intersystem dependencies facilitates more rapid reconfiguration of systems to meet operational requirements.

4.1.3.1.1 Layers

Layers facilitate making software intensive systems interoperable in the presence of rapidly changing technologies and/or multiple technology choices.

Layering makes it possible to design a system of systems that has technology independence, scalability, decentralized operation, appropriate architecture, and supporting standards, security, and flexibility. Layering can also accommodate heterogeneity, accounting, and cost recovery. Excellent examples of layering include the use of TCP/IP to decouple communications link technologies from applications that use communications and the use of hypertext transport protocol (HTTP) and hypertext mark-up language (HTML) to separate presentation from storage and retrieval functions.

Middleware provides an example of the layering principle. It separates the applications from the operating systems on which the applications run [].

Middleware services are sets of distributed software that exist between the application and the operating system and network services on a system node in the network.

By decreasing the dependence of applications on a particular operating system, middleware increases the ease of moving applications to new computers or systems and decreases

dependence on operating systems that might fall out of favour in the commercial marketplace.

The Common Object Request Broker Architecture (CORBA) is one of a number of Middleware platforms for distributed systems development [OMG 98]. Other platforms include Microsoft's .NET and Java 2 Platform, Enterprise Edition.

4.1.3.1.2 Standards

An essential aspect of architecture is the establishment of technical standards. In general, standards define common elements such as user interfaces, system interfaces, representations of data, protocols for the exchange of data, and interfaces accessing data or system functions.

Technical standards provide a number of advantages for the systems architect. With regard to interoperability, standards are important because multiple vendors accept them. Thus standards increase the likelihood that a collection of systems from diverse sources will be able to interoperate. It has become generally accepted by now that although standards are certainly beneficial, simple adherence to standards is not sufficient to guarantee interoperability.

Even when there are accepted standards and compliant products, interoperability is facilitated but not assured as there are options within standards and different releases and versions of products.

Finally, it is important to realize that technical standards are, by themselves, necessarily incomplete from the standpoint of a system or component designer. The operational scenarios that a system is expected to support play an integral role. This range of scenarios defines the context in which a system is to perform specific desired functions and thus provides a meaningful reference for testing and evaluation.

4.1.4 Data Interoperability

Experience suggests that left to their own devices, the designers of individual systems will often make locally optimal decisions about data definitions and formats.

Data formats resulting from such local decisions may not be compatible when operational requirements dictate that a network of systems be called upon to interoperate. Thus architectural design must provide guidance to developers to minimize the applications-layer incompatibilities that inevitably arise when systems with different purposes must communicate with each other.

Examples of approaches to data interoperability include

- Single data definition for all systems. This approach can be problematic when applied on a large scale to a complex, evolving system or system of systems. The task of agreeing on definitions consumes a great deal of effort and time that might be better used elsewhere. Also, when a single set of definitions is mandated for all applications, definitions are no longer locally optimal, and thus such mandates often encounter substantial resistance in implementation.
- Object orientation. This is a technically common approach for developing data definitions by encapsulating the internal details of the data.
- Middleware. Common middleware (for example CORBA, J2EE) can be used to help achieve data interoperability.
- Extensible data model. This approach uses an extensible data model and standardized interface. The Simple Network Management Protocol is an example.
- Extensible Mark-up Language (XML). This approach requires agreement on the contents and meaning of the XML schema. Thereafter, application data is

communicated in XML that conforms to the schema. Like single data definition, obtaining agreement on the schema can be difficult but XML also promises extensibility of data mark-up.

Legacy systems, which have been built around frequently unique data definitions, pose a major challenge to interoperability. Industry has developed a number of approaches by which systems not originally designed for interoperability can interoperate to exchange information (including the data “bus” approach, the data dictionary approach, the data translator approach, and the data server approach).

4.1.5 Architecture Styles for Addressing Interoperability

Shaw defines an architectural style as a family of architectures constrained by component/connector vocabulary, topology, semantic constraints and analysis methods supported [Shaw 95]. Styles can be different because they use different vocabularies, topologies or constraints. The vocabulary of component/connector is frequently the principal aspect characterizing the style.

Choosing an architectural style for a system is usually the earliest of the early design decisions.

The styles are similar to design patterns but covering larger chunks of architecture, capturing a recurring solution to a recurring problem. Styles are usually ambiguous about the number of components involved. Styles may be ambiguous about the mechanism(s) by which the components interact although some styles bind this explicitly. Styles are always ambiguous about the function of the system: one of the components may be a database, for example, but the kind of data may vary.

Diverse researches have proposed in the literature several collections of architectural styles to be used in software intensive systems [Shaw 96] [Buschmann 96] [Bass 98].

Here are some of the architectural styles described that are more adequate to solve the interoperability problem. This description will especially deal with data centred architectures and object oriented architectures.

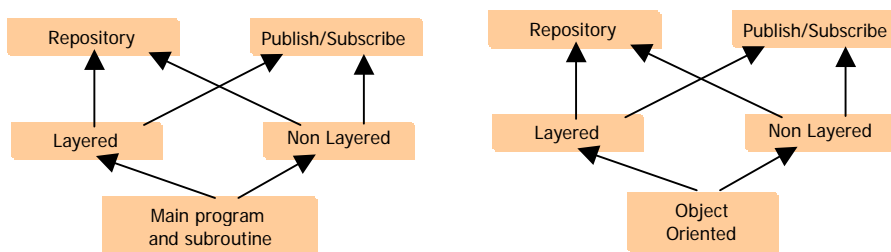


Figure 9: 1 Architectural Styles addressing interoperability

- Data Centred.

Data centred architectures have the goal of achieving data integration. Typically data centred architectures are composed by client elements and a shared data element. A client runs on an independent thread of control. The shared data that all client elements access may be a passive repository (such as a file) or an active repository (such as a blackboard). A blackboard sends notification to subscribers when data of interest changes, and thus is active.

- Call and Return.

These architectures have the goal of achieving the qualities of Maintainability and scalability. Call and return architecture are specialized in other styles as described in Figure 9. Main program and subroutine architecture is the classical programming paradigm. The goal is to decompose a program into smaller pieces to help to achieve maintainability. A program is decomposed hierarchically. There

is typically a single thread of control and each component in the hierarchy gets this control from this parent and passes it along its children. Remote procedure call systems are main program and subroutine systems that are decomposed into parts that live on computers connected via a network. The goal is to increase efficiency by distributing the computations and taking advantage of multiple processors

The object-oriented paradigm emphasizes the building of data and the knowledge of how to manipulate and access that data. Furthermore, the object-oriented paradigm clearly defines the responsibilities and the collaboration of the different classes. Access to the object is allowed only through provided operations.

Layered systems are ones in which building elements are assigned to layers to control intercomponent interaction. In the pure version of this substyle, each level communicates only with its immediate neighbours. The goal is to achieve the qualities of interoperability, maintainability and portability.

In the following chapter this style, solutions adopted and the implications for ATM systems will be described.

4.1.5.1 Data Centred Architectures

Many real-time applications have a requirement to model some of their communication patterns as a pure data-centric exchange, where applications publish (supply or stream) "data" which is then available to the remote applications that are interested in it. Relevant real-time applications can be found in Air Traffic Management, industrial automation, distributed control and simulation, telecom equipment control, sensor networks, and network management systems. More generally, any application requiring (selective) information dissemination is a candidate for a data-driven network architecture.

Predictable distribution of data with minimal overhead is of primary concern to these real-time applications. Since it is not feasible to infinitely extend the needed resources, it is important to be able to specify the available resources and provide policies that allow the middleware to align the resources to the most critical requirements. This necessity translates into the ability to control Quality of Service (QoS) properties that affect predictability, overhead, and resource utilization.

The need to scale to hundreds of publishers and subscribers in a robust manner is also an important requirement. This is actually not only a requirement of scalability but also a requirement of flexibility: on many of these systems, applications are added with no need/possibility to reconstruct the whole system. Data-centric communications decouples senders from receivers; the less coupled the publishers and the subscribers are, the easier these extensions become.

Distributed shared memory is a classic model that provides data-centric exchanges. However, this model is difficult to implement efficiently over a network and does not offer the required scalability and flexibility. Therefore, another model, the **Data-Centric Publish-Subscribe (DCPS)** model, has become popular in many real-time applications.

This model builds on the concept of a "global data space" that is accessible to all interested applications. Applications that want to contribute information to this data space declare their intent to become "Publishers." Similarly, applications that want to access portions of this data space declare their intent to become "Subscribers." Each time a Publisher posts new data into this "global data space," the middleware propagates the information to all interested Subscribers.

Underlying any data-centric publish subscribe system is a *data model*. This model defines the "global data space" and specifies how Publishers and Subscribers refer to portions of this space. The data-model can be as simple as a set of unrelated *data structures*, each identified by a *topic* and a *type*. The topic provides an identifier that uniquely identifies some

data items within the global data space. The type provides structural information needed to tell the middleware how to manipulate the data and also allows the middleware to provide a level of type safety. However, the target applications often require a higher-level data model that allows expression of aggregation and coherence relationships among data elements.

Another common need is a **Data Local Reconstruction Layer (DLRL)** that automatically reconstructs the data locally from the updates and allows the application to access the data 'as if' it were local. In that case, the middleware not only propagates the information to all interested subscribers but also updates a local copy of the information.

There are commercially available products that implement DCPS fully and the DLRL partially (among them, NDDS from Real-Time Innovations and Splice from THALES Naval Nederland); however, these products are proprietary and do not offer standardized interfaces and behaviour that would allow portability of the applications built upon them.

ATC systems are systems that help ATC Controllers fulfil their mission. These systems are usually organized around data that represent all the flights that are in the air, have just landed, or should take-off soon. In aggregate, these data are commonly called "FlightPlans" and are quite complex for they gather:

- The initial description of the plan (flight identification, aircraft identification, aircraft description, airline identification, place of departure, place of arrival, scheduled times...),
- The route description (i.e., the list of points and timestamps that the aircraft is supposed to follow and then is following).

And more generally, all the events that have affected the flight during its lifetime (e.g., controllers identification, clearances, controllers orders...).

Some of those items are static or almost static (e.g., the aircraft identification), but others are dynamic (e.g., the route description may change). Upon changes, actions have to be undertaken quickly (e.g., HMI visualization, activation of the conflict detection...). A pure client/server design where each part that needs information gets the information by a synchronous call would be unlikely to meet the requirement of fast reaction upon change.

Different parts that act on the data are not concerned by all of them (e.g., the Trajectory Predictor is only concerned with the route description). Given that there are many data items, a simple design where the whole FlightPlan is broadcast whenever there is a change within would simply result in an unaffordable amount of data transfer.

On the other hand, data items that constitute a FlightPlan, cannot be seen as unrelated atomic pieces of data. For instance a geographical point is a whole and you need to get all the coordinates as an atomic item (just the first one is meaningless). In addition, some controllers operations result in changes of several items that must be seen as an atomic change.

4.2 Maintainability Tactics

Maintainability is about the capability of the software product to be modified. More specifically, it addresses analysability, changeability, stability and testability. In the context of OATA we concentrate on the changeability and stability subcharacteristics.

4.2.1 Tactics for Localizing Expected Modifications

The responsibilities that are assigned to modules greatly influence the cost of making a change. Depending on how the assignment was done, a specific change can affect either a single module or multiple ones. The goal of this set of tactics is to directly affect as few modules as possible with a single change by presenting guidelines for how responsibilities are assigned.

The tactics for localizing expected modifications include

- *Maintain semantic coherence.* Semantic coherence refers to the relationships between a module's responsibilities. Semantically coherent responsibilities are related by what they do, carrying out the same or at least similar functions.
- *Isolate the expected change.* Separating the responsibilities that are likely to change from those that are unlikely to change separates architecture into fixed and variant parts. This enables more design effort to be devoted to making a selected subset of modules as easy to change as possible.
- *Raise the abstraction level.* Raising the abstraction level, and thus making a module more general, allows that module to calculate a broader range of functions based on input.
- *Limit options.* Limiting the set of possible modifications will reduce the variations that need to be considered in the design and simplify constructing a system suitable for modification.
- *Abstract common services in the primary modules.* Localize services that are used commonly by a variety of consumers.

4.2.2 Tactics for Restricting the Visibility of Responsibilities

If a module is affected by a change, it is important to know whether that change will become visible outside the module. If it will, changes to other modules will most likely be required.

The tactics for restricting visibility include

- *Hide information.* This tactic is based on dividing a module's responsibilities into two categories: public and private. Public responsibilities are those that are visible from both inside and outside the module. Private responsibilities are those that are visible only from inside the modules. The goal of this tactic is to limit the public responsibilities and make them visible through an interface.
- *Maintain existing interfaces.* This tactic is based on keeping interfaces constant across a particular change. That is, the module developer will maintain the old identity, syntax, and semantics of an existing interface even if the modification changes them.
- *Separate the interface from the implementation.* This tactic allows the realization of the implementation later in the development process than the interface specification.

All three tactics will not be applicable to OATA. The tactics *Hide Information* and *Separate the interface from the implementation* are part of the set of "good practices" enforced by the OATA architecture team. The tactic *maintain system interfaces* is not relevant to OATA, but it is kept for the sake of completeness.

4.2.3 Tactics for Preventing the Ripple Effect

A ripple effect from a modification is the necessity for making changes to modules that are not directly affected by that modification. This necessity occurs because of a dependency between the module that is directly affected and another module that is dependent on it.

The tactics that can be used in the OATA logical architecture for preventing a ripple effect include:

- *Break the dependency chain.* This tactic refers to the use of an intermediary to keep one module from being dependent on another and therefore to break the dependency chain. Typically, the name of the intermediary depends on the type of dependency it breaks. The following examples of intermediaries can also be seen as tactics:

- *Maintain the independence between Platform Independent Model (PIM) and the Platform Specific Model (PSM):* This is the approach chosen and applied by the OATA architecture development team.
- *Use a publish-subscribe pattern.* A publish-subscribe pattern breaks the dependency of a data consumer on the identity of the data producer.
- *Use a repository.* A repository can be used to break two types of dependencies:
 4. The dependence of a data consumer on the identity of the publisher (as in the Publish-Subscribe pattern).
 5. The dependency of the data consumer on the data's syntax. Modern repositories allow the consumer to specify the type in which data is presented to them, regardless of the data's type in the repository.
- *Limit communication paths.* Restricting the modules with which a given module will communicate has the effect of ensuring that no superfluous dependencies exist.

Even though the following tactics are not applicable to OATA -because they are more related to possible platform implementations- they are presented for completeness:

- *Break the dependency chain:*
 - *Use a name server.* A name server breaks a dependency on the runtime location of a module.
 - *Use a virtual machine.* Virtual machines break dependencies on computations specific to a particular situation. Examples of virtual machines include the hardware abstraction layer and the Factory pattern [Gamma 95].
 - *Use a dynamic-scheduling algorithm.* Some scheduling algorithms, such as semantic importance- based scheduling, guarantee that deadlines will be achieved within certain restrictions.
- *Make the data self-identifying.* Tagging the data with identification information, such as sequence number (as in network protocols), syntax descriptions (as in some languages with dynamic runtime typing), or identity (as in free-form parameter invocation), will break dependencies on either sequencing or syntax.

4.3 Efficiency Control Tactics

These two tactics help the software architect to control the efficiency of the system as it is being developed and throughout its life cycle.

Efficiency control tactics are not relevant to the OATA logical architecture. The document presents them in order to have a complete overview of the different tactics that are applicable to a system's architecture.

4.3.1 Efficiency Requirements Tactic

This tactic refers to the definition of specific, quantitative, measurable efficiency requirements for efficiency scenarios.

Efficiency requirements control efficiency by explicitly stating the required efficiency rigorously enough so that the architecture evaluator can quantitatively determine whether or not the architecture meets that requirement. If the architect does not know where he or she is going, it doesn't matter how he or she goes. But if he or she can quantify where he or she needs to be, then the alternatives can be evaluated and it is possible to select the best way of meeting non-functional requirements related to efficiency.

4.3.2 Instrumenting Tactic

Instrument the system architecture as you build it to enable measurement and evaluation of workload scenarios, resource requirements and efficiency requirements compliance. This tactic is related to inserting code probes at key points to enable the measurement of pertinent execution characteristics. For example, probes may record the number of times each efficiency validation scenario executes, the end-to-end response time, the number of transactions in each session, the number of I/O operations and so on. The validator needs information on resource requirements for critical portions of code, not just the total for the software. To collect this data, the developers must insert code to call system-timing routines, and write key events and relevant data to files for later analysis.

4.4 Independent Tactics

These four tactics improve the efficiency of the system by reducing its computer resource requirements. They are called independent tactics because they can be applied independently, they do not conflict.

These tactics are related to the PSM and thus are not relevant to the OATA logical architecture. The document presents them in order to have a complete overview of the different tactics that are applicable to a system's architecture.

4.4.1 Centring Tactic

The purpose of the centring tactic is to identify the dominant workload functions and minimize their processing. Centring leverages efficiency by focusing attention on the parts of the software architecture that have the greatest impact on performance. This tactic is derived for the well-known "80-20 rule" for the execution of code within a program. This observation states that 20% or less of a program's code accounts for 80% or more of its computer resource usage. We extend the code execution rule to apply to the demand for system functions.

Centring is concerned with identifying the subset (the 20% or less) of the system functions that will be used the most (80% or more) of the time. These frequently used functions are the dominant workload functions. These dominant workload functions also cause a subset ($\leq 20\%$) of the operations in a software system to be executed most ($\geq 80\%$), as well as the code within operations, and so on. Improvements made in these dominant workload functions thus have a significant impact on the overall performance of the system.

For object-oriented architectures as the OATA logical architecture, centring involves identifying the critical use cases. These are critical use cases to the operation of the system, or that are important to responsiveness as seen by a user. Critical use cases may also include those that relate to risks involving efficiency. The centring tactic is so important that identifying critical use cases is a step of the performance validation process described in the document "Description of the OATA architecture of non-functional requirements validation process".

4.4.2 Fixing-Point Tactic

The fixing-point tactic emphasizes that for responsiveness, fixing should establish connections at the earliest point in time, such that retaining the connection is cost effective.

Fixing connects the desired action to the instructions used to accomplish that action, or the desired result to a data used to produce it.

The fixing point is a point in time. The latest the fixing point is during execution, just before the instructions are to be executed. Dynamic binding in object oriented languages, or polymorphic function calls that cannot be resolved during compilation, exhibit late fixing. Fixing can also establish the connection at a several possible earlier times: earlier in the execution, at system initialization, during compilation, or even outside the software.

In some cases, early fixing may reduce the flexibility of the design.

An example of early versus late fixing would be the implementation of an “Ordered Collection” class. The late-fixing solution would be to add items to the collection as the new items arrive, and then perform a sort when it is accessed. Early fixing would insert the item in the proper location, or sort the collection each time an item is added. The best solution here depends on the environment in which the collection is to be used; in particular how often the user needs the ordered versus the unordered data.

4.4.3 Locality Tactic

The locality tactic promotes the creation of actions, functions and results that are close to the physical computer resources. For example a list of names needs to be sorted, and that data is locally stored in memory instead of being on disk on a remote node in the network, the locality is good. The types of locality are:

- spatial, as in the above example
- temporal (i.e., time)
- effectual (i.e., purpose or intent)
- degree (i.e., intensity or size)

The types of locality are spatial, as in the above example, temporal (i.e., time), effectual (i.e., purpose or intent), and degree (i.e., intensity or size).

In an object oriented architecture as the OATA logical architecture, it is important to keep related data and behaviour together. An object should have most of the data that it needs to make a decision or perform an action. Objects that have very frequent interactions should be assigned to the same processor, and should perhaps even to be compiled and linked together.

4.4.4 Processing Versus Frequency Tactic

The processing versus frequency tactic states that the product of processing times frequency should be minimized. This tactic is concerned with the amount of work done in processing a request and the number of requests received. It seeks to make a trade-off between the two. It may be possible to reduce the number of requests by doing more work per request, or vice versa.

4.5 Synergistic Tactics

The following three tactics are known as synergistic tactics because they improve the overall efficiency of a system via cooperation of processes competing for computer resources.

The synergistic tactics depend on cooperation to reduce delays for resource contention. If all objects do not cooperate, the desired improvement is not achieved.

These tactics are not relevant to the OATA logical architecture. The document presents them in order to have a complete overview of the different tactics that are applicable to a system's architecture.

4.5.1 Shared Resources Tactic

The shared resources tactic is recommending sharing resources when possible. When exclusive access is required, it is important to minimize the sum of the waiting time and the servicing time.

Suppose that flight plans records need to be updated. To avoid corrupting the data, exclusive access is required until the record has been updated. One way to obtain exclusive access for the update is to lock the entire flight plans database. This approach minimizes servicing time, it requires less overhead to lock the entire database than it does to lock an individual flight

plan record because we only need to check the status of one lock indicator. This approach maximizes waiting time, however, because no other process can access the database until the lock has been released. Another approach is to lock individual flight plan records. This approach minimizes waiting time because other processes can access other records, but maximizes servicing time because there is a separate lock status indicator for each record. The best alternative is likely to be a compromise that locks a group of records and thus minimizes the sum of waiting and servicing time. In general, the appropriate solution will depend on the access patterns for the individual application.

4.5.2 Parallel Processing Tactic

The parallel processing tactic is the recognition of a trade-off: Execute processing in parallel (only) when the processing speed up offsets communication overhead and the resource contention delays.

The concurrency may be real concurrency or apparent concurrency. In real concurrency, the processes execute simultaneously in different processors. In this case, the processing time is reduced by an amount proportional to the number of processors. In apparent concurrency, processes are multiplexed on a single processor. Here the situation is more complicated. While some of the processing may be overlapped, each process will sometimes experience additional wait time due to contention for the same resource such as a disk.

4.5.3 Spread-The-Load Tactic

The spread-the-load tactic is: spread the load when possible by processing conflicting loads at different times or in different places.

This tactic is similar to the shared resources tactic, they both address resource contention delay. The shared resources tactic reduces the delay by minimizing waiting time and servicing time. This tactic reduces the delay by reducing the number of processes that need a resource at a given time and by reducing the amount of resource that they need. In some cases there may be some overlap between the two tactics. For example by reducing the amount of the resource that concurrent processors need, we are also reducing waiting time.

Evaluating spread-the-load alternatives requires constructing and solving performance engineering models to quantify resource contention delays for each alternative.

4.6 Patterns

Using available experience in the elaboration and implementation of logical architectures, it is possible to identify three basic patterns that are almost always implicated in interoperability issues: Translator, Controller, and Extender. Figure 10 presents them together with taxonomy of their possible usage [Keshav 98].

A Translator converts data and functions between module formats, without changing the content of the information. It does not need knowledge of where the data came from or where it is sent.

A Controller coordinates and mediates the movement of information between modules using some predefined decision-making process. The Controller needs to know the identities of the modules for which decisions are made.

An Extender adds new features and functionality, therefore enhancing module capability. Whether or not the Extender needs to know the identity of the modules with which it interacts depends on the specific application.

The following sections present more detail regarding the definition of these patterns.

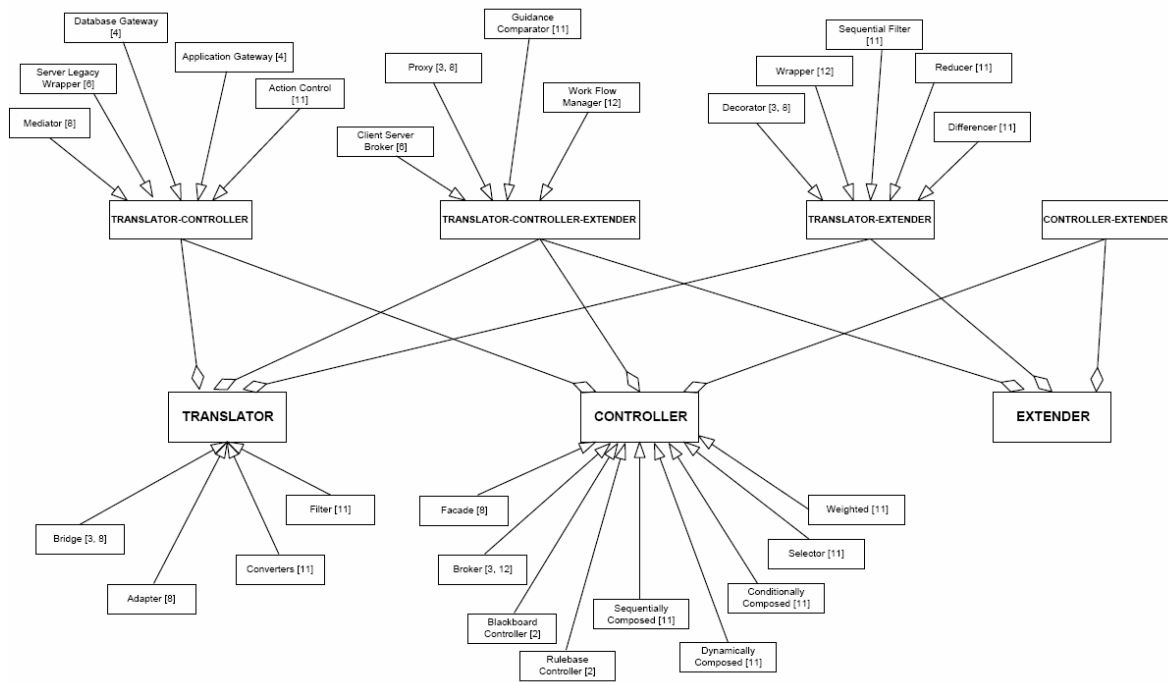


Figure 10: Interoperability patterns Taxonomy

4.6.1 Translator Patterns

A translator pattern has three elements:

- I/O ports.
- Input domain/output range.
- Converter relationship.

To be considered a translator pattern, each of these elements must meet the following requirements:

6. I/O Ports.

The translator may receive data³ from and send data to the ports of architectural components without knowing the identities of those components.

7. Input domain/output range.

The input domain formed by data from the translator's input ports has a uniform structure and/or content that is predefined by the functionality of the translator. The output range confines the results of the translation.

8. Converter relation.

A converter is a mathematical relation that performs the translation of the input domain to the output range.

Figure 11 presents a schematic view of a translator patterns.

³ In this discussion, "data" includes procedure calls, signals, parameters, event calls, shared data, remote procedure call and information.

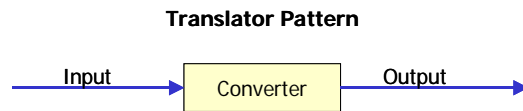


Figure 11: Translator Patterns Schema

4.6.2 Controller Patterns

A controller patterns coordinates and mediates the movement of information between architectural components using a predefined decision-making processes that includes determining:

- Which data is passed.
- From which architectural components to accept data.
- To which architectural components to send data.
- A combination of the above decisions.

The controller pattern maintains the information in its original incoming format (format changes imply the patterns translator). The following requirements characterise a controller patterns:

9. *I/O Ports.*

10. *Decision making strategies.*

The controller must have at least one strategy or rule on which it bases its decisions. It may also have a set of decision-making strategies from which to choose one or a combination of strategies to be applied at once.

11. *Decision making algorithm.*

A controller has a decision-making algorithm that determines the strategy to be employed and applies it accordingly. The strategies may be static, or dynamic (changing throughout their execution). These strategies may be stored internally or externally with respect to the controller. The useable strategies may change according to the data, components contributing or receiving data and or state of computation.

Figure 12 shows a schema for the Controller Patterns

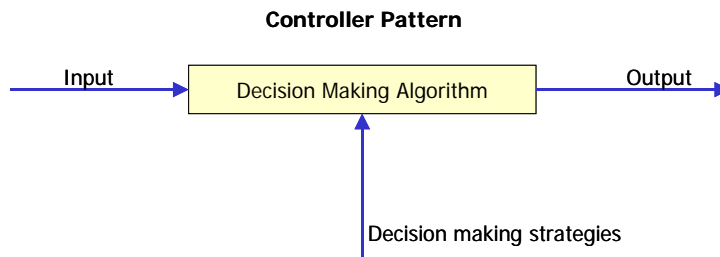


Figure 12: Controller Patterns Schema

4.6.3 Extender Patterns

An extender patterns adds new features and functionality to a component to adapt its behaviour. An extender patterns may also add functionality to the system as a whole to enhance its overall capability. The extender patterns complete the behaviours not described in either the Translator or the Controller patterns. An example of an extender patterns is the addition of a component in “front” of a server or a client to perform mundane tasks such as opening files and security check. Figure 13 presents a schema for this pattern.

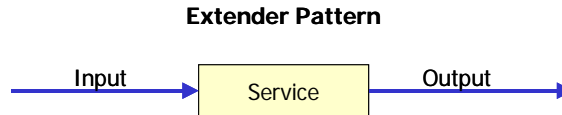


Figure 13: Extender Patterns Schema

Seven broad categories of interoperability conflict resolution are defined. Each category represents an integration sub-architecture (a piece of the overall solution) used to resolve interoperability conflicts [Davis 00].

- **Variance** – all conflicts that necessitate hiding the implementation of the components involved fall under this category. A translator such as a bridge or adapter would be implemented to resolve a transparency problem.
- **Coordination** – all conflicts that require only a decision-making strategy, be it directing data or choosing the next component to execute, comprise this category. The implementation of a controller such as a façade or a selector resolves the determination problem.
- **Deficiency** – all conflicts that have need of some device to perform tasks not covered by translation or control fall in this category. Extension such as, security checking, use of a buffer or repository, provides the additional functionality to resolve these conflicts.
- **Coordination with Variance** – all conflicts that are in need of interface transparency and decisive routing are encompassed by this category. The elements of translation and control are combined to fix arbitration conflicts, as in a mediator [BMRSS96] or an application gateway.
- **Coordination with Deficiency** – all conflicts needing determination or direction plus some means or device to store and exchange information. A shared repository using a controller to direct the information and an extender to store it may resolve this type of conflict.
- **Variance with Deficiency** – all conflicts having both transparent implementation needs and requiring additional services for seamless integration are contained in this category. Translation joined with extension provides for this functionality, e.g. a proxy.
- **Coordination with Variance and Deficiency** – all conflicts that demand arbitration plus must use resources to allow communication to flow without problems are harboured in this category. To gain the full functionality, a combined translation, control and extension mechanism must be created as a manager. This can be found in the Broker pattern that underlies middleware such as CORBA.

5 APPENDIX B: SEMANTIC INTEROPERABILITY

5.1 Introduction

Even though semantic interoperability will not be an issue for OATA (because OATA is proposing the use of the logical architecture to ensure semantic interoperability with legacy systems), this chapter is included to ensure that the issue is understood. Furthermore, this information could be used during the transition from legacy to the OATA logical architecture.

Semantic interoperability tries to ensure that the exchange of services and data among the elements of different systems makes sense, i.e. that requesters and providers have a common understanding of the meaning of the requested services and data. Semantic interoperability is based on agreements on, for example, the algorithms for computing the requested values, the side effects of procedures, or the source or accuracy of requested data elements.

Semantic interoperability problems are related to the difficulty of identifying semantically equivalent information that is used in different systems. Having different meanings for the same logical components, or for the metamodel components can result in the subjective mapping of structures, overlapping conflicts, incompatibility of data structures, data precision inconsistencies, or incomplete information. All of these issues result in general in the lack of interoperability between systems. In the instance of the OATA logical architecture the semantic interoperability problem will probably be exacerbated by the need to incorporate different existing legacy systems, each with the own meaning and domain terms.

From a practical point of view, it is almost impossible to assess the semantic interoperability of two systems. It should cover not only operational semantics and behavioural specs of components, but also agreements on names, context-sensitive information, agreements on concepts (ontologies) —which moves names agreements one level up in the meta-data chain—, non-functional requirements, and so forth. This is caused by the need to identify the different “meanings” of all the different terms used by both systems. To address this problem, system architects use an “ontology” or formal conceptualisation of the “architecture”. This ontology is used to specify a set of constraints that declare what should hold in any possible system related to the “architecture”. Thus, a legal “architecture” description is an architecture that satisfies the ontology constraints. From this point of view semantic interoperability assessing the degree of matching between the ontology and architecture would perform validation.

Since there is no OATA ontology available, the description will focus on the description of the basic rules and concepts behind the ontology.

5.2 Elaboration of an Ontology

The following paragraphs present a brief summary of the process that should be followed to elaborate ontology.

12. Formalisation of concepts

There are several ways to describe an object. These ways are characterised either by the attributes (or adjectives), their parts (or nouns) or their functions (or verbs), or by a combination of all or part of them. The role of the concept formalisation is to identify the different ways in which an object is described and to produce a structured representation of it.

A concept model should identify the relations between the different objects in such a way that each concept should have an associated set of constraints that declare what should be true in any possible environment in which the concept might be used.

Even though there are several methods to produce this formalisation, all of them have in common the following “logics”.

- a. Use of concept expressions (formulae with one free variable) for the description of object sets
- b. Use of axioms to define relations between the concepts
 - i. Subsumption
 - ii. Equivalence
 - iii. Disjointness

The use of these logics allows defining any concept in terms of its meaning in different worlds or contexts.

13. Information Integration

The purpose of this step is to model complex relations that map onto concepts in the global view as simple rule sets. This method could also be used to model constraints between different information sources.

The integration can be reduced to three activities [Visser 03]:

- a. Query Answering: Does the instantiation of the goal predicate follow from the ontology, the mapping rules and the constraints?
- b. Query Containment: Does the body of a query expression follow from another query expression given ontology, mappings and constraints?
- c. Use maximally contained re-writings to find the most general query using only source predicates that are contained in a given query.