 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Interoperability Specification (IOS) – V1
D601.021

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Interoperability Specifications (IOS) – V1
Deliverable No.	D601.021
Dissemination Level	PU
Nature	R
Document Version	1.0
Date	2014-05-30
Contact	Frédéric Loiret
Organization	OFFIS
Phone	+49 441 9722 481
E-Mail	Frederic.Loiret@offis.de

AUTHORS TABLE

Name	Company	E-Mail
Frédéric Loiret	OFFIS	Frederic.Loiret@offis.de
Gray Bachelor	IBM	gray_bachelor@uk.ibm.com
Rainer Ersch	Siemens AG	rainer.ersch@siemens.com
Andreas Keis	Airbus Group	Andreas.k.Keis@eads.net

REVIEWERS TABLE

Name	Company	E-Mail
Christian El-Salloum	AVL	christian.elsalloum@avl.com
Jean-Luc Johnson	Airbus Group	Jean-Luc.Johnson@eads.com
Rubén de Juan Marín	ITI	rjuan@iti.es

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.1	18-03-2014	Initial version.	
0.2	29-03-2014	Table of content refined.	
1.0	30-04-2014	Final version.	

CONTENT

1 INTRODUCTION 7

1.1 PREAMBLE 7
1.2 ROLE OF DELIVERABLE 7
1.3 RELATIONSHIP TO OTHER CRYSTAL DOCUMENTS 8
1.4 STRUCTURE OF THIS DOCUMENT 8

2 IOS PREREQUISITES 9

2.1 IOS & RTP OVERVIEW 9
2.2 IOS ADOPTION HISTORY 10
2.3 BASIC TERMINOLOGY 11
2.3.1 Interoperability Scenarios 11
2.3.2 Engineering Artefacts and Lifecycle Artefacts 12
2.3.3 Adapters and Service Consumers/Providers 12
2.4 CRYSTAL IOS LAYERED ARCHITECTURE 13
2.4.1 IOS Positioning with OSLC 14
2.4.2 IOS Positioning with other existing Engineering Standards 15
2.4.3 Scope of the IOS V1 and Subsequent Versions 15
2.5 EXAMPLES OF CONCRETE SCENARIOS 16

3 IOS V1 BASIC LIFECYCLE INTEROPERABILITY SCENARIOS 17

3.1 OVERVIEW OF THE SCENARIOS 17
3.2 DESCRIPTIONS OF THE SCENARIOS 17
3.2.1 Interoperability Pattern 1: Access remote artefact 18
3.2.2 Interoperability Pattern 2: Store link to remote artefact 19
3.2.3 Interoperability Pattern 3: Access remote artefact with UI redirection 19

4 INTEROPERABILITY SPECIFICATIONS V1 22

4.1 CORE CONCEPTS AND PRINCIPLES INHERITED FROM OSLC 22
4.1.1 IOS Vocabulary 22
4.1.2 Resources 22
4.1.3 Linktypes 22
4.1.4 Properties 22
4.2 NOTES ON APPLICATION OF OSLC V2.0 AT IOS V1.0 22
4.3 IOS CHAPTER – CORE CAPABILITIES 22
4.3.1 Common Core Capabilities 22
4.3.2 Service Provider Capabilities 23
4.3.3 Service Consumer Capabilities 24
4.3.4 Advanced Core Capabilities 24
4.4 IOS CHAPTER – DOMAIN SUPPORT: REQUIREMENT MANAGEMENT 25
4.5 IOS CHAPTER – DOMAIN SUPPORT: ARCHITECTURE MANAGEMENT 25
4.6 IOS CHAPTER – DOMAIN SUPPORT: ASSET MANAGEMENT 26
4.7 IOS CHAPTER – DOMAIN SUPPORT: CHANGE REQUEST MANAGEMENT 26
4.8 IOS CHAPTER – DOMAIN SUPPORT: QUALITY MANAGEMENT 27

5 CRYSTAL ENGINEERING METHODS TO IOS SERVICES 29

5.1 ENGINEERING METHOD (EM) TO IOS SERVICES MAPPING METHODOLOGY 29
5.2 ENGINEERING METHOD STEPS – IOS SERVICE MAPPING TABLE 29
5.3 ENGINEERING METHOD STEPS – IOS SERVICE MAPPING TABLE EXAMPLE 30

6 EXTENSION MECHANISMS OF IOS V1 31

6.1 CRYSTAL EXTENSIONS 31

6.2 TOWARDS IOS SUSTAINABILITY 31
7 TERMS, ABBREVIATIONS AND DEFINITIONS 33
8 REFERENCES 34
9 ANNEX 35

Content of Tables

Figure 1: RTP & RTP Instances 10
Figure 2: Lifecycle Artefacts and Engineering Artefacts....12
Figure 3: The CRYSTAL IOS Layered Architecture. 14
Figure 4: IOS Interoperability Pattern 1: Access remote Lifecycle Artefact 18
Figure 5: IOS Interoperability Pattern 1: Access remote artefact - Sequence diagram 18
Figure 6: IOS Interoperability Pattern 2: Store link to remote artefact 19
Figure 7: IOS Interoperability Pattern 2: Store link to remote artefact - Sequence diagram....19
Figure 8: IOS Interoperability Pattern 3: Access remote artefact with UI redirection 20
Figure 9: IOS Interoperability Pattern 2: Access remote artefact using a delegated UI - Sequence diagram .20
Figure 10: Example of an EM – IOS Service Mapping table (excerpt)....30

Content of Figures

Table 1: Terms, Abbreviations and Definitions....33
Table 2: Examples of Concrete Scenarios related to "Lifecycle Interoperability" and "other Interoperability Topics"....35

1 Introduction

1.1 Preamble

CRYSTAL aims at fostering Europe's leading edge position in embedded systems engineering in particular regarding quality and cost effectiveness of safety-critical embedded systems and architecture platforms. However, because of the heterogeneity and complexity of such systems covered by the project, requiring multiple engineering competences across various engineering disciplines, their developments become a huge challenge to overcome by European developing organizations, notably because of:

- The heterogeneity of engineering tools & data involved in their development platforms across the development lifecycle (encompassing requirements engineering, design, wide range of V&V activities from dynamic testing, formal analysis, fault-tree analysis, etc.),
- The increasing need in the context of safety-critical systems to bridge the gap between development platforms and operational ones (e.g., for including human in-the-loop or virtual testing, heterogeneous co-simulation, or for monitoring and maintaining large-scale distributed applications), with the goal to improve the development and decision-making processes in large developing organizations,
- The distributed and multi-tiers nature of development teams in nowadays' large European organizations, spread over multiple countries and suppliers.

In order to overcome these challenges, CRYSTAL leverages on a momentum initiated by past and on-going ARTEMIS projects[1] around a common vision for the Establishment of Recognized International Open Standards of Lifecycle Tool & Data Integration Platforms for Systems Engineering (or "*Tool Chains*"). The main idea of the so-called Interoperability Specifications (IOS) initiated in these projects is to rely on common interoperability services, providing a common ground for integrating lifecycle and engineering tools across different engineering disciplines and from multiple stakeholders involved in the development of large scale safety-critical systems (e.g., requirements engineers, developers, V&V experts, but also business analysts and managers). The common denominator of the IOS across the projects, and among the CRYSTAL's partners, is based on a lightweight and domain-agnostic approach, providing basic capabilities for handling the whole lifecycle of engineering artefacts manipulated throughout the development of safety-critical embedded systems, and is presented in this deliverable.

1.2 Role of deliverable

This deliverable presents a first outcome of extensive technical discussions that have been conducted in the first year of the project among representatives of all CRYSTAL's stakeholders (i.e., end-users from aerospace, automotive, healthcare and rail domains, tool and integration solution providers, and technology brick providers from SP6), in order to reach a first level of consensus regarding the shape, scope, and content of the CRYSTAL IOS, and based on the approach that has been initially proposed, adopted and refined in the ARTEMIS projects CESAR, iFEST and MBAT, and on which CRYSTAL use case owners put a clear focus.

Therefore, this deliverable has to be considered as a first baseline for finalizing such a consensus in the project, and for presenting directions on how the CRYSTAL IOS will be extended in the subsequent versions of it (by inputs from other ARTEMIS projects such as MBAT, and of course from within CRYSTAL, as

[1] In particular CESAR, iFEST and MBAT.

outcomes of its *Technical Management Process* that has been deployed and kick-started across its work packages structure).

Moreover, the content of the specification presented in this deliverable ought to be used as a first reference document for implementing basic IOS-compliant CRYSTAL bricks focused on lifecycle interoperability scenarios, already covering a large part of the V-model, from requirements engineering to architecture and quality management, including change request management, and traceability throughout the development process. The current version of the specification presented in the deliverable can be already applied to cover significant parts of basic lifecycle interoperability needs already elicited by the CRYSTAL Use Cases in their *Engineering Methods*.

1.3 Relationship to other CRYSTAL Documents

This document is related to the deliverable D601.031 - Report on Standardisation Work - V1.

1.4 Structure of this document

This deliverable is structured as follows:

- The **section 2** provides the prerequisites for comprehending the main concepts and principles behind the CRYSTAL *Interoperability Specification* (IOS), and presents its layered architecture used as a reference conceptual model in the project for characterizing the scope and content of the IOS.
- The **section 3** presents the basic and generic lifecycle interoperability scenarios and patterns from the CRYSTAL use cases that are already supported by the IOS concepts V1 defined in this document.
- The **section 4** defines, from a detailed technical standpoint, the Interoperability Specification V1, focused on Lifecycle Interoperability so far, and consisting of so-called *IOS Chapters*.
- The **section 5** gives an overview of the part of the *CRYSTAL Technical Management Process* that has been deployed within the project which is focused on the mapping of CRYSTAL *Engineering Methods* (or workflows) onto IOS V1 concepts and services defined in this deliverable.
- Finally, the **section 6** gives a brief overview of the extension process of the CRYSTAL IOS, which outcomes will be presented in the subsequent versions of this deliverable, and aiming at paving the way towards IOS sustainability beyond the project.

2 IOS Prerequisites

2.1 IOS & RTP Overview

The IOS – as it has been defined by former ARTEMIS projects (see next section, *IOS Adoption History*) -- consists of a specification for achieving common Tool & Data Interoperability in heterogeneous Systems Engineering development environments. In particular, it encompasses the specification of three main aspects:

- The specification of communication paradigms and protocols to be used for exchanging information between integrated Tools and Data repositories,
- The specification of data exchange formats (or *syntax*, referring to the formats used for serializing data as strings, e.g., RDF/XML, XMI/XML, JSON, etc.), and
- The specification of the semantics of the information to be interpreted and exchanged across these Tools and Data repositories (or *abstract syntax*, referring to the definition of sets of concepts for lifecycle integration, defined with their properties and relationships).

In general, the CRYSTAL Interoperability Specification (IOS):

- Relies on Common Interoperability Principles,
- Is based on existing Interoperability Standards,
- Is based on, and related to, other relevant Interoperability & Engineering Standards,
- Is open (i.e., IOS is royalty-free use),
- Is extensible,
- Aims to be widely accepted by industrial users and tool vendors.

In particular, the CRYSTAL IOS (as described in detail section 2.4) is focused on:

- *Lifecycle Interoperability*, and on
- *Other Interoperability Topics for supporting In Depth Systems Engineering Activities*.

The part of the IOS focused on *Lifecycle Interoperability* is based on Artefacts used in Systems Engineering Development Environments ("*Tool Chains*"):

- To support collaboration between stakeholders of different roles, different engineering disciplines, and different industrial domains,
- To enable status reporting, traceability, dashboarding, analysis, prediction, data collection for reports, automation support, etc., between Tools and Data Repositories.

The key idea pushed forward in the IOS consists in relying on standardized integration interfaces for supporting lifecycle interoperability, with the goal to overcome redundant integration problems (e.g., related to point-to-point and ad-hoc integration architectures, isolated tool silos, users locked-in, reusability & maintenance of integration assets) across the boundaries of engineering disciplines, application domains and tool providers. Such standardized integration interfaces have to define lightweight and generic concepts as a common denominator for all the artefacts used holistically throughout the development cycle of CRYSTAL Use Cases. In the context of lifecycle interoperability, the main focus is put on the semantics of the links and dependencies between the artefacts crossing the boundaries between the engineering disciplines (i.e., related to requirement engineering, design & implementation, and V&V related activities).

The part of the IOS focused on *Other Interoperability Topics for supporting In Depth Systems Engineering Activities* will be based on existing Engineering Standards identified as already widely used among CRYSTAL partners and European developing organizations. Indeed, CRYSTAL aims at supporting

engineering activities focused on end-users' businesses, and which require detailed, specific and "bespoke" semantics and methodologies besides lifecycle interoperability. Such engineering activities encompass heterogeneous co-simulation, combination of dynamic testing and formal static analysis, variability management, design space exploration, just to name a few.

It is worth noting that by nature, activities related to lifecycle interoperability on the one hand, and those related to in depth Systems Engineering on the other hand are entangled. It is therefore in the scope of CRYSTAL to clearly elaborate on the complementarities and areas of overlap between these two dimensions of interoperability in order to fully fulfil integration requirements and needs from the CRYSTAL Use Cases.

The concept of a European *Reference Technology Platform* (RTP) has also emerged within the ARTEMIS eco-system (initially introduced in the CESAR project). Basically, the main assets of the RTP are the following:

- A library of ready-to-integrate Engineering Tools, with their respective *adapters* implementing IOS-compliant integration interfaces,
- A set of Integration Software Development Kits (SDKs) and dedicated Support Tools (e.g., CRYSTAL Platform Builder related tools) for specifying, implementing, instantiating, tailoring, deploying and maintaining *RTP instances* (or "*Tool Chains*"),
- A set of Methodologies and guidelines for supporting advanced lifecycle interoperability scenarios and in depth Systems Engineering activities by the CRYSTAL technical work packages.

The Figure 1 presents a graphical representation of the *RTP* and *RTP instances*. The latter consists of an integrated subset of RTP bricks based on the IOS.

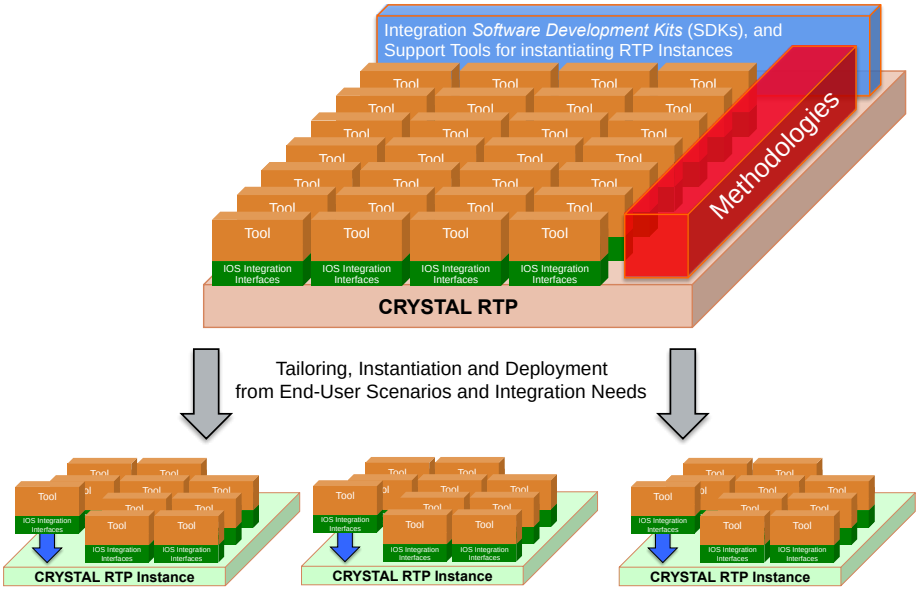

Figure 1: RTP & RTP Instances

2.2 IOS Adoption History

Embedded systems are getting more and more complex and the different engineering disciplines and stakeholders are using more and more highly specialized methods and tools to execute their tasks. Unfortunately, most of those methods and tools manage their information in isolated environments and keep them behind private APIs. Attempts to bring all information together in one single, universal repository to

connect them better typically failed because the universal repositories do not provide enough functionality for the highly specialized tasks. In addition, new innovative approaches to optimize tasks in the engineering process, because they cannot be introduced due to the inflexibility of such centralized environments.

Therefore, the importance of Interoperability in Engineering Environments is rising and it is becoming more and more crucial for successful and efficient systems engineering. The CESAR project[2] introduced the idea of a "*Reference Technology Platform*" (RTP), where the different parts where connected based on a standardized *Interoperability Specification* (IOS).

After some investigations in different interoperability technologies, the CESAR project has selected the emerging open standard OSLC[3] (*Open Services for Lifecycle Collaboration*) as basis for the CESAR Interoperability Specification IOS. Other projects (iFEST[4], MBAT[5], and now CRYSTAL) followed this approach. The iFEST project came in an independent evaluation to the same conclusion that OSLC is the right solution for their interoperability needs. MBAT continued with the IOS foundation laid by CESAR and adopted it for their "Combined Model-based Analysis and Testing of Embedded Systems" methodologies. CRYSTAL has now taken over with writing the story further.

In the meantime, the OSLC open initiative has grown up from a "loosely coupled" web community, to a member section of the open standard organization OASIS[6]. Many commercial and open source products have adopted the open standard and the number of participating organizations is constantly growing. The ARTEMIS projects are very well connected with the OSLC standard organization though key project members serving as OSLC Steering Committee members and workgroup leads. Although OSLC is already an excellent basis for the CRYSTAL IOS, the project has already identified some additional needs for interoperability in their use cases, which will most likely lead to enhancements of the OSLC standard and an extension of the CRYSTAL IOS by other standards.

2.3 Basic Terminology

In this section are simply given definitions of the most important terms to be grasped by the readers as a prerequisite for understanding the basic concepts underlying the IOS.

2.3.1 Interoperability Scenarios

In the context of Systems Engineering development environments, an *Interoperability Scenario* elicits a workflow for which each activity explicitly captures an exchange of information between stakeholders, tools, and data repositories, and in particular across the boundaries of engineering disciplines. These activities are either performed manually or automatically. Elicitations of Interoperability Scenarios with the characterization of information to be exchanged for each activity (e.g., types of artefacts to be exchanged) and by which means (e.g., via browsing or graphical representations of artefacts, via manual or automatic notification mechanisms, etc.), constitute the main inputs to be considered for defining the IOS and its extensions.

The CRYSTAL *Engineering Methods* can be considered as elicitations of Interoperability Scenarios (but it is worth noting that in some cases, some activities defined by the end-users are not directly related to interoperability per se, but are focused on engineering activities performed internally from within a tool without any need of exchanging information beyond its scope).

[2] http://www.cesarproject.eu
[3] http://open-services.net
[4] http://www.artemis-ifest.eu
[5] http://www.mbat-artemis.eu
[6] http://oasis-oslc.org

2.3.2 Engineering Artefacts and Lifecycle Artefacts

The Figure 2 below illustrates two integrated engineering tools (or RTP Bricks) with their *adapters* (see the definition of the latter in the next subsection, an adapter basically consists of *Service Providers* and/or *Service Consumers*, only the providers are represented on Figure 2), and introduces the notions of *Lifecycle Artefact* and *Engineering Artefact*.

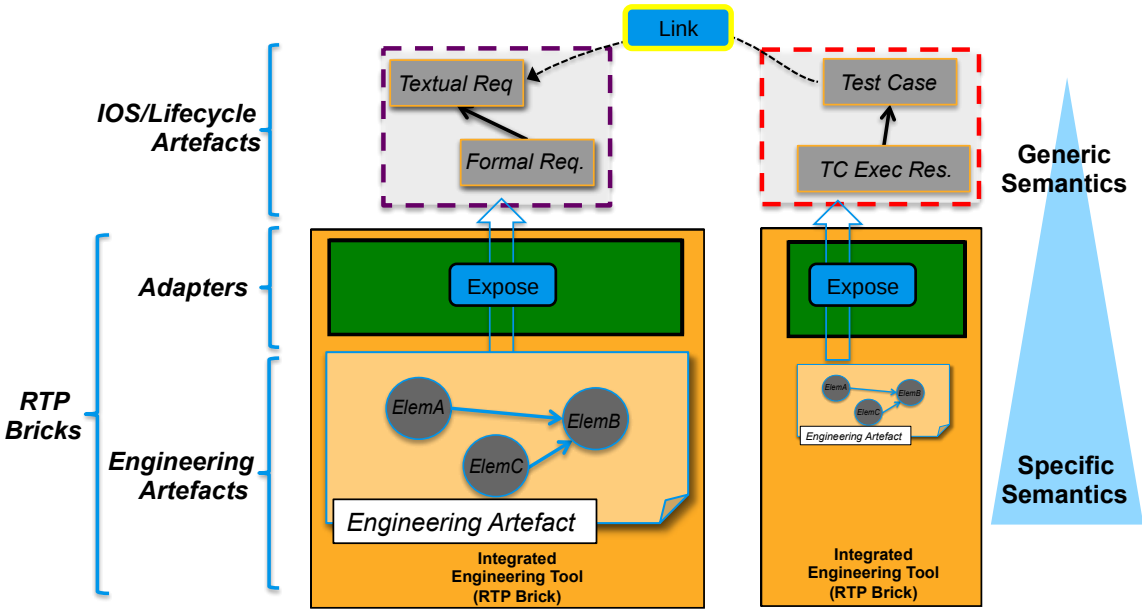

Figure 2: Lifecycle Artefacts and Engineering Artefacts.

Basically, *Engineering Artefacts* constitute the assets used internally within the Engineering Tools (e.g., as files or as data chucks stored in a database). Their semantics and syntaxes are defined by standard formats (e.g., UML/XMI or standard C and executable files) or by proprietary formats (e.g., MDL files used within Simulink). An Engineering Artefact is generally defined as a container and namespace of Artefact *Elements* (e.g. a set of classes stored in a UML class diagram).

Engineering Artefacts and their Elements are mapped onto *Lifecycle Artefacts*, the latter being the first class entities used for supporting Lifecycle Interoperability scenarios. The syntax and semantics of Lifecycle Artefacts, the basic services defined for manipulating them (read and write access), and the communication protocols used for serializing them between integrated tools and data repositories are defined by the IOS, as presented in section 2.4.

2.3.3 Adapters and Service Consumers/Providers

As already introduced above, an *Adapter* consists of *Service Consumers* and *Providers*.

As described in the IOS layered architecture (see section 2.4), major parts of the IOS will be based on RDF/XML and HTTP & Core Web Technologies. The preferred architectural style, to be used with the IOS, is the so-called Representational state transfer (REST[7]). One basic concept of this architectural style is the separation of concerns in *Clients* and *Servers*. This means clients do not have direct access to data stored (or "owned") by a server, but the server "provides" services which can be "consumed" by the client to read, query and manipulate the data. In the IOS we call the servers providing services "Providers" and the client consuming services "Consumers".

[7] For a detailed description of REST, *see* http://en.wikipedia.org/wiki/Representational_state_transfer

Typical services are:

C.R.U.D Services:

- **Create** a resource using HTTP POST and content being resource format of choice,
- **Read** a resource using HTTP GET and standard HTTP content negotiation,
- **Update** a resource using HTTP GET to get resource properties to be updated and HTTP PUT to send updated resource,
- **Link** a resource using properties where values are just URIs (*Unified Resource Identifiers*),
- **Delete** a resource using HTTP DELETE.

Querying for resources

- Query Capability, which has a base URI,
- Clients form query URI and HTTP GET the results,

UI Preview & Delegation

- Rich hover:
  Scenario supported: hover over link to get in context preview of resource
- Resource Delegation:
  Simple resource format defined and retrieved using HTTP content negotiation
  (e.g. Resource Picker for links, or Creation Factory)

Many of these services are defined in the specification of the OSLC standard. They are clustered by so called (engineering) domains: Change (Request) Management, Requirements Management, Quality Management, Architecture Management and others. Common principles are defined in the Core Specification (presented in section 4).

Ideally, the tools, used to build the Engineering Environments for the CRYSTAL Use Cases, would provide these services out of the box. As a matter of fact, there are already some commercial and open source products which have IOS compliant service interfaces and some of the CRYSTAL partners have agree to build such services for their products. In these cases the tools with the out of the box service interfaces would be already suitable bricks for the RTP library. If a tool does not have an IOS compliant interface, an adapter can be built, containing a web component to provide the needed IOS service(s) and uses internally the proprietary APIs to connect to the backend of the application. The tool together with the IOS adapter will then constitute an RTP brick. Where necessary, the brick providers in the CRYSTAL project are responsible for building the adapters for the tools needed for the CRYSTAL Use Cases.

2.4 CRYSTAL IOS Layered Architecture

The Figure 3 presents the IOS layered architecture, which has been enhanced in CRYSTAL from the CESAR and the MBAT IOS. The top part of the figure encompasses the tool and domain-specific syntax and semantics, possibly based on proprietary and island solutions, which is basically out-of-scope of the IOS. On the contrary, the bottom part sketches the scope of the IOS, (a) specifying a common way for handling Lifecycle Interoperability (with respect to communication protocols, syntax, services and semantics used as a common ground for exchanging lifecycle artefacts and control flows between integrated engineering tools in a standardized way), and (b), the set of other Engineering/Interoperability Standards, supporting in depth Systems Engineering activities, and to be interfaced with Lifecycle Interoperability concepts.

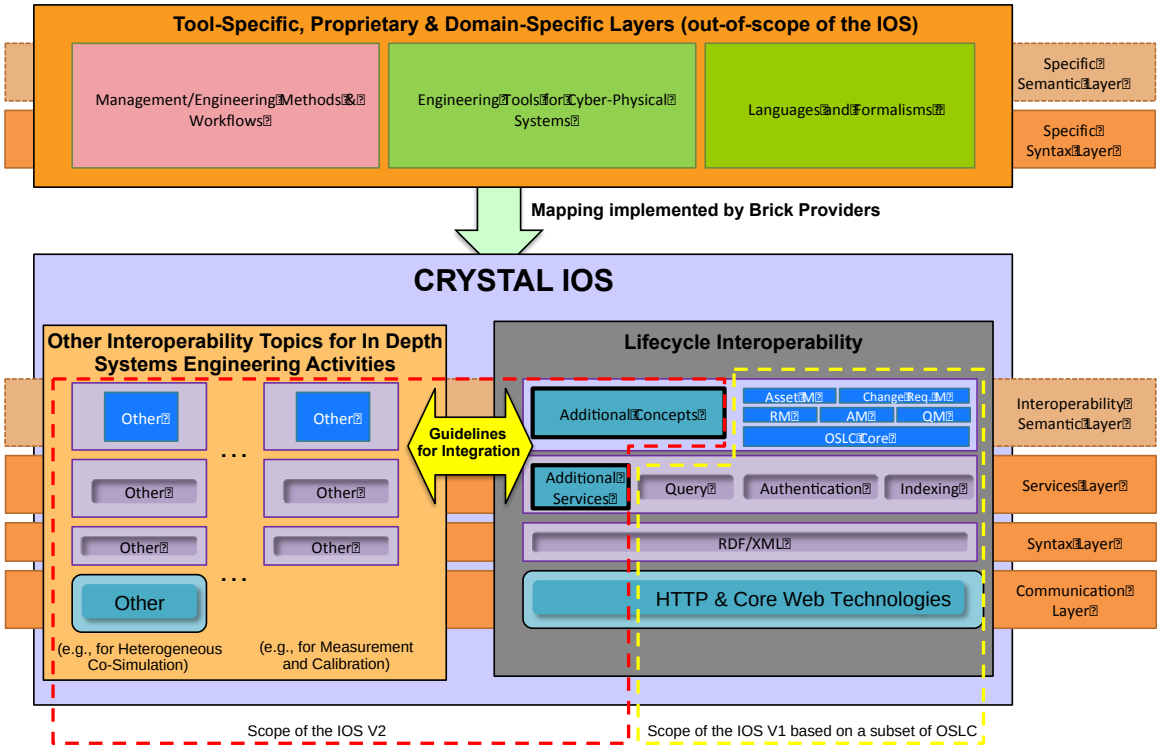

Figure 3: The CRYSTAL IOS Layered Architecture.

2.4.1 IOS Positioning with OSLC

The Lifecycle Interoperability related part of the CRYSTAL IOS is based on a subset of OSLC (*Open Services for Lifecycle Collaboration*[8]). OSLC defines a set of specifications[9] focusing on the support of lifecycle activities, but only a subset of these specifications have been assessed to be part of the CRYSTAL IOS so far, and are the following:

- **OSLC Core specification** (see detail in section 4.1), relying on HTTP and core Web technologies, defining standardized way for encoding and representing lifecycle artefacts, defining basic services for creating, requesting, updating, and deleting artefacts,
- **OSLC Requirement Management specification** (see detail in section 4.4), defining primary requirement lifecycle artefacts,
- **OSLC Architecture Management specification** (see detail in section 4.5), defining primary architecture lifecycle artefacts,
- **OSLC Quality Management specification** (see detail in section 4.8), defining primary lifecycle artefacts related to testing activities,
- **OSLC Change Request Management specification** (see detail in section 4.7), defining primary lifecycle artefacts for handling change requests across the engineering disciplines,
- **OSLC Asset Management specifications** (see detail in section 4.6), mainly used in the context of CRYSTAL for defining a standardized way for wrapping *Engineering Artefacts* (i.e., assets used internally within the engineering tools as described in section 2.3.2).

[8] http://open-services.net
[9] http://open-services.net/specifications/

These OSLC specifications have been considered to be integral part of the IOS for lifecycle interoperability after evaluations conducted on CRYSTAL Engineering Methods and across the Domains (i.e., from SP2 to SP5). En example of such an assessment is presented in section 5.

2.4.2 IOS Positioning with other existing Engineering Standards

Besides Lifecycle Interoperability, the CRYSTAL IOS will also cover other interoperability topics for system engineering activities and will include relevant specifications and standards. Examples for topics that have been identified in the use cases are:

- Functional Mock-up Interface (FMI) for co-simulation,
- ASAM ODS for measurement data,
- Semantic-preserving model transformations between input and output models,
- ReqIF to interchange requirements,
- AUTOSAR specifications in the automotive sector.

In several scenarios these standards are employed independently of each other, *but in many cases a conceptual link has to be established between such standards and OSLC for Lifecycle Interoperability*. The IOS will define specifications to bridge such standards and guidelines for their integration.

Proposals to integrate a standard in the IOS can come from each use case owner. They are first evaluated by WP6.1, and the final decisions will be made by the Technical Board of CRYSTAL.

The focus of the Interoperability Specification V1 is on Lifecycle Interoperability, therefore the concrete content for this section will be provided in the subsequent versions of this deliverable.

2.4.3 Scope of the IOS V1 and Subsequent Versions

The Figure 3 also illustrates the scope of the content of this first version of the CRYSTAL Interoperability Specifications ("*IOS V1*", presented in detail in section 0). The current version is then exclusively focused on Lifecycle Interoperability, and based on a subset of the existing OSLC specifications. In the subsequent versions of this deliverable (i.e., in D601.022 and D601.023, respectively "*IOS V2*" and "*IOS V3*", to be released at M24 and M36), significant enhancements of the IOS will be compiled, notably via:

- Support of other Engineering Standards as mentioned in section 2.4.2, for which areas of overlap with the concepts defined for Lifecycle Interoperability will be clearly identified (in particular via "*Guidelines for Integration*"),
- Extensions of the current Lifecycle Interoperability specifications (as represented on Figure 3 by the boxes "*Additional Concepts*" and "*Additional Services*"). These extensions will come from the following inputs:
    - From IOS extensions that have been already adopted in ARTEMIS projects, notably from iFEST and MBAT (e.g., MBAT proposed IOS lifecycle interoperability extensions for supporting tight combination of testing and analysis methodologies),
    - From other existing OSLC specifications that will be evaluated in CRYSTAL (e.g., related to support for automation, reporting or configuration management),
    - And most importantly from CRYSTAL SP6 technical work packages (i.e., WP6.3 to WP6.13) and CRYSTAL domain ontology work packages (i.e., WP2.9, WP3.8, WP4.7 and WP5.4, respectively addressing ontologies for the aerospace, automotive, healthcare and rail domains). The cross-WP activities for defining these extensions are integral part of the so-called CRYSTAL *Technical Management Process*, and are driven by the Engineering Methods and Requirement elicited by the CRYSTAL use cases.

2.5 Examples of Concrete Scenarios

For illustrative purposes, simple examples of scenarios focused on Lifecycle Interoperability and on other Interoperability Topics (based on, and related to other Engineering Standards) are given in Annex I (page 35).

3 IOS V1 Basic Lifecycle Interoperability Scenarios

3.1 Overview of the Scenarios

As already detailed around the Figure 3 presenting the CRYSTAL IOS layered architecture, this first version of the deliverable is focused on the support of Lifecycle Interoperability activities, and the current content of the IOS consists of the OSLC Core specifications defining the cornerstone principles of the IOS, and of the OSLC specifications for Requirement, Architecture, Quality Management (respectively OSLC RM, AM and QM), and the OSLC Change Request Management and Asset Management specifications.

This set of specifications can be notably used for supporting the following generic lifecycle interoperability operations and scenarios:

1. The Creation, Request, Update and Delete (CRUD) operations to be performed on Lifecycle Artefacts (from OSLC Core),
2. The creation of links, and navigation between and across Lifecycle Artefacts in a systematic way (from the W3C Linked Data principle, part of OSLC Core),
3. The redirection of User Interfaces (UI) for basic query or creation of remote Lifecycle Artefacts, or for displaying user friendly graphical representations of Engineering Artefacts across the engineering disciplines and stakeholders (from OSLC Core),
4. Included here are examples of the application of these basic services patterns across the domains as defined today by OSLC:
   a) The exposition of basic Lifecycle Artefacts from different engineering phases (from OSLC RM, AM and QM),
   b) The lifecycle scenario for supporting basic change requests (from OSLC Change Request Management),
   c) The scenarios for which Engineering Artefacts (i.e., not IOS/OSLC based) have to be exposed as “raw data/file” (from OSLC Asset Management).

3.2 Descriptions of the Scenarios

The basic multiple brick interoperability scenarios covered here use the following convention:

- Brick A is a “local” tool which originates the transaction from either a User Interface (UI) or a program event. Local meaning as initiating or having primary control of processing or receiving the results of remote processing, or being the place that flow returns to after some secondary processing.
- Brick B is a “remote” tool which receives and responds to the transaction. Remote meaning as directed or secondary processing or receiving temporary control of processing or parallel processing or relinquishing control to another tool.
- Brick C is an “intermediate” or “additional” tool needed to complete the transaction, e.g. it may be a specific UI component of say Brick A or B but its purpose is to enable Web compliant information rendering (i.e. via HTTP and HTML) in response to say a GET.

Bricks can have local, or remote, or distributed functions, e.g., thick client, thin client, browser, and server components. No distinction is made whether tools have user interfaces.

Bricks, in IOS V1 use services to communicate and so may use infrastructure components like Web or proxy servers, these are out of scope (transparent) for the basic tool interoperability scenarios.

The IOS V1 specification addresses the external interfacing needed for tool interoperability and not the enabling infrastructure, like co-existence of operating systems or protocol conversion.

These schemes aim to indicate recurring patterns of interoperability and do not show any needed authentication of users or services, nor the internal interfaces required to add or enable service-based transaction, nor do they address licensing needs.

3.2.1 Interoperability Pattern 1: Access remote artefact

Within Brick A access a remote artefacts for Create, Read, Update or Delete (CRUD) in remote Brick B.

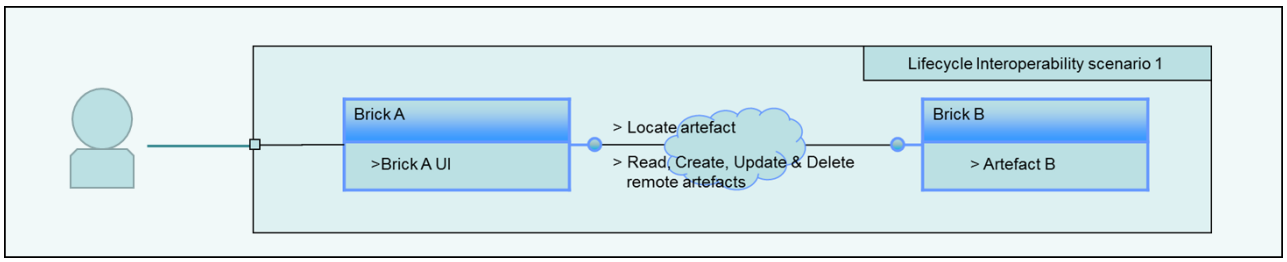

Figure 4: IOS Interoperability Pattern 1: Access remote Lifecycle Artefact

Note: Implementers need to ensure the idempotency (i.e., the result of the request should be independent of the number of times the service is executed) of HTTP operations on remote artefacts, particularly PUT and POST.

An overview of the main transactions to achieve this scenario follows on the Figure below.

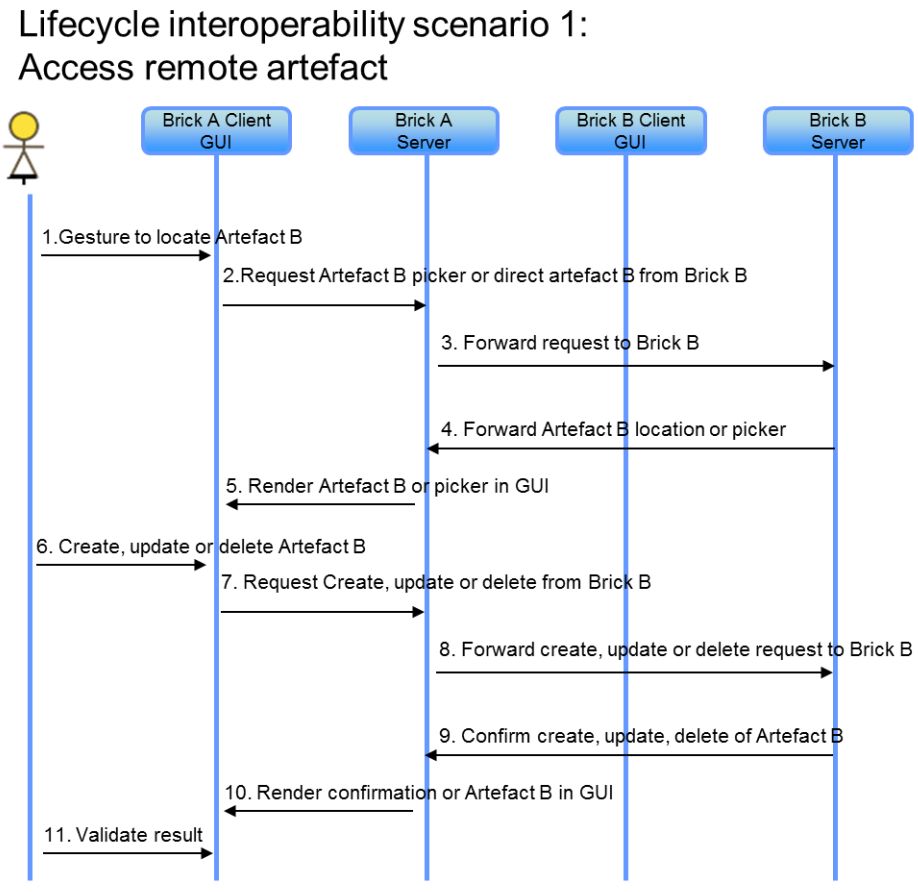

Figure 5: IOS Interoperability Pattern 1: Access remote artefact - Sequence diagram

3.2.2 Interoperability Pattern 2: Store link to remote artefact

Within Brick A saves (holds) a local link (URI) to a remote artefact owned by Brick B

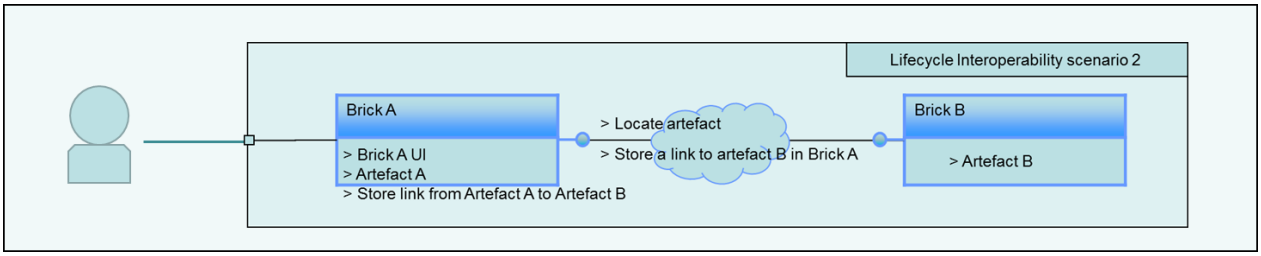

Figure 6: IOS Interoperability Pattern 2: Store link to remote artefact

It is assumed that the URI is used to associate Artefact A with Artefact B, it is also possible that Brick A just registers the URI in a catalogue.

An overview of the main transactions to achieve this scenario follows on the Figure below.

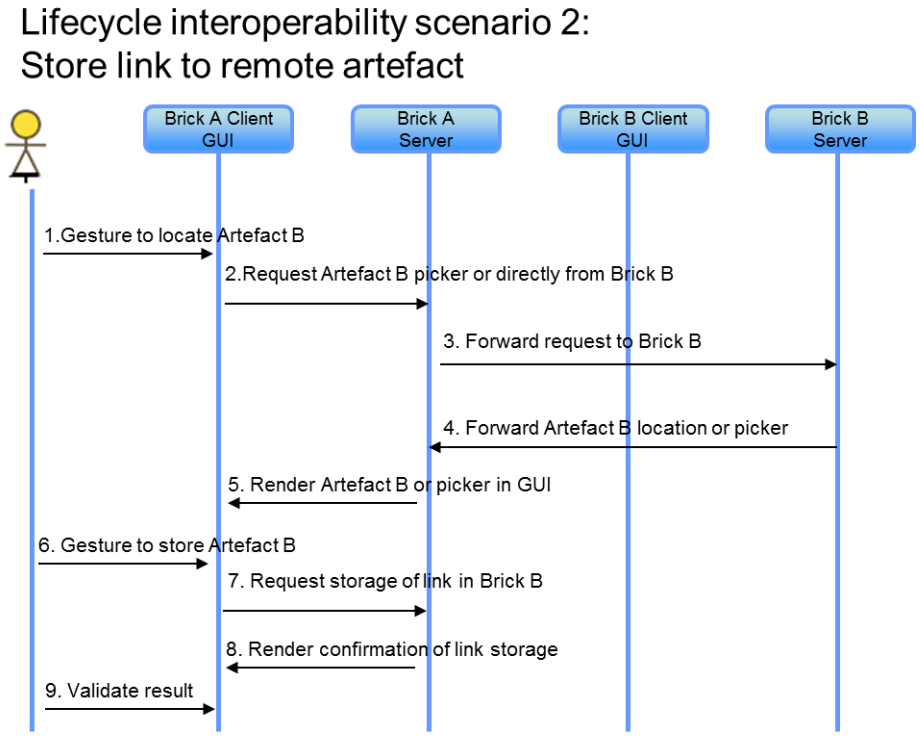

Figure 7: IOS Interoperability Pattern 2: Store link to remote artefact - Sequence diagram

3.2.3 Interoperability Pattern 3: Access remote artefact with UI redirection

Redirected user interface for remote artefact to Brick C, e.g., a browser.

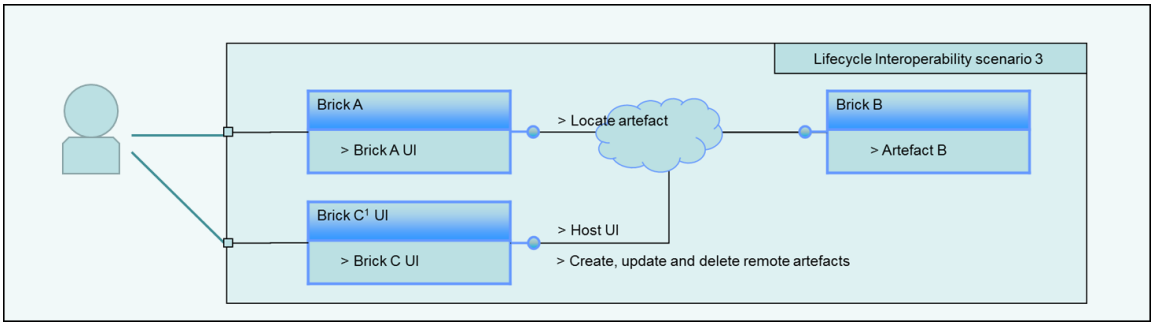

Figure 8: IOS Interoperability Pattern 3: Access remote artefact with UI redirection

Note: Brick C is a separate UI component. It can be hosted by Brick A, B or a separate Brick C. An example is an OSLC delegated UI which is Brick B UI hosted by Brick A.

An overview of the main transactions to achieve this scenario follows on the Figure below.

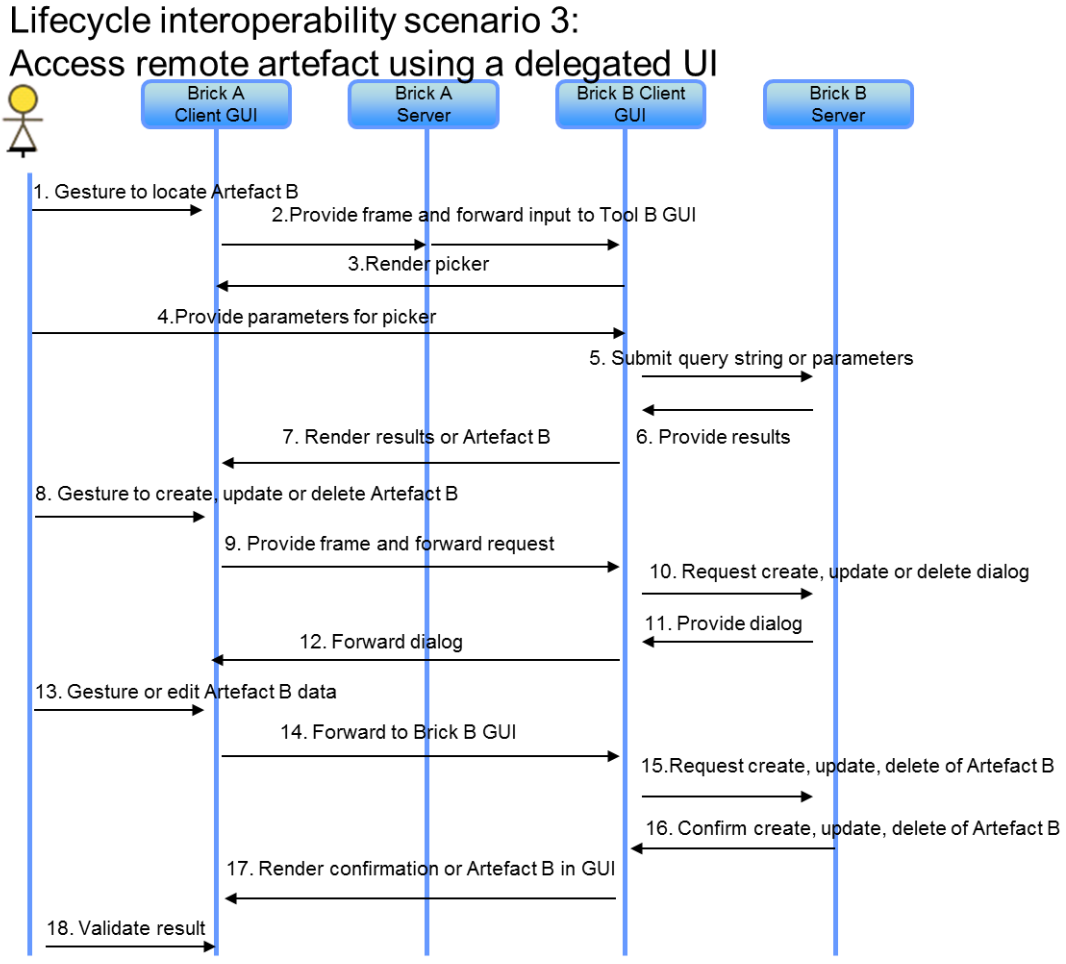

Figure 9: IOS Interoperability Pattern 2: Access remote artefact using a delegated UI - Sequence diagram

Within scenario 3 the IOS provides multiple options such as query and creation dialogs and delegated user interfaces. A delegated UI removes the burden from Brick A of processing data that is likely to be incompatible with its own native data.

These basic scenarios are tailored by the use of domain vocabulary such as for:

1. Exposing of Lifecycle Artefacts from different engineering phases like Requirements, model Resources and Test Plans (using OSLC RM, AM and QM respectively),
2. Support for change requests (OSLC Change Management). Note here that there is no implied lifecycle state control; any consumer wishing to change the state of a resource is assumed to do so by a POST of a new status property/value pair to a Change Request resource.
3. Support where Artefacts are exposed without reference to one of the lifecycle domain directly (e.g. not IOS/OSLC based) examples could be part of a file system used by a tool which is not available as a Brick “raw data/file” (using the OSLC Asset Management spec).

4 Interoperability Specifications V1

4.1 Core Concepts and Principles inherited from OSLC

At V1 the CRYSTAL IOS is concerned with the following tool interoperability styles:

- Service based tool interoperability used WWW and loosely coupled principles.

CRYSTAL IOS adopts:

- Service based tool interoperability from OASIS[10], which defines service as "a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description[11]."
- RESTful[12] Web service technologies.

4.1.1 IOS Vocabulary

Crystal IOS V1.0 adopts an information model that is defined through OSLC V2.0 domains. The IOS vocabulary is defined with the domain specifications selected from OSLC V2.0.

4.1.2 Resources

The various defined Resources are detailed below with the domains.

4.1.3 Linktypes

The various defined linktypes are defined below with the domains

4.1.4 Properties

The various defined properties are defined below with the domains.

4.2 Notes on application of OSLC V2.0 at IOS V1.0

There are certain logs of issues with the OSLC V2.0 that implementers should also refer to, these are noted. It is recommended that implementers follow, review and join the OASIS OSLC working group proceedings around these issues to draw upon the community experience.

4.3 IOS Chapter – Core Capabilities

4.3.1 Common Core Capabilities

IOS ref.	Capability	Description	Source Org	Source ref	Notes
1	Service entry point	Uniform Resource Identifier (URI)	W3C/IETF	http://tools.ietf.org/html/rfc3986	

[10] http://en.wikipedia.org/wiki/OASIS_%28organization%29
[11] http://en.wikipedia.org/wiki/Service_%28systems_architecture%29 - cite_note-1
[12] http://en.wikipedia.org/wiki/Representational_State_Transfer

2	Service protocol	Hypertext Transfer Protocol (HTTP)	W3C/IETF	http://tools.ietf.org/html/rfc2616	
3	Resource definition & operations	Resource Description Framework	OSLC W3C LDP	http://open-services.net/bin/view/Main/OslcCoreSpecification?sortcol=table;table=up - OSLC_Defined_Resources http://www.w3.org/TR/PR-rdf-syntax/ http://www.w3.org/2000/01/rdf-schema - >	
4	Service document types	RDF/XML	W3C	http://www.w3.org/TR/REC-rdf-syntax/ http://www.w3.org/TR/2014/REC-turtle-20140225/	
5	Error responses	HTTP status codes	W3C	http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html	
6	Namespace extension		W3C	http://www.w3.org/TR/REC-xml-names/	
7	Paging	Resource Paging	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification?sortcol=table;up= - Resource_Paging http://open-services.net/resources/tutorials/oslc-primer/resource-paging/	
8	Simple query	A simplified and specific query	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification#Query_Capabilities	Non SPARQL query

4.3.2 Service Provider Capabilities

IOS ref.	Capability	Description	Source Org	Source ref	Notes
9	Discovery	Engage a provider to understand service content available	OSLC	http://open-services.net/wiki/core/Discovery-3.0/	
10	Service provider catalogue	Service Provider	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification?sortcol=table;up= - Service_Provider_Resources	
11	Resource	Create, Read, Update	OSLC	http://open-	PUT should

	CRUD services	and Delete Resource through GET, PUT POST and DELETE transactions		services.net/wiki/reconciliation/OSLC-Reconciliation-Specification-Version-2.0/	not be assumed to be idempotent
12	Creation Factory	A service to create new resources	OSLC		
13	Dialogs	User interaction dialog	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification	See resource: Dialog
14	Delegated User Interface (DUI)	In-context information displayed from a link	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification?sortcol=table;table=up;up= - Delegated_User_Interface_Dialogs http://open-services.net/wiki/core/User-Interface-Previews-3.0/	
15	UI preview	User interface or resource preview	OSLC	http://open-services.net/bin/view/Main/OslcCoreUiPreview	

4.3.3 Service Consumer Capabilities

IOS ref.	Capability	Description	Source Org	Source ref	Notes
16	Provider registration	A consumer client registers a Service provider (URI and description)	OSLC	http://open-services.net/resources/tutorials/integrating-products-with-oslc/implementing-an-oslc-provider/providing-service-providers-and-catalogs/	Tutorial
17	Provider authentication	A consumer can authenticate successfully with a Service Provider	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification?sortcol=table;up= - Authentication	
18	Service selection	A consumer can locate a service type from a provider	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification	See Overview
19	Delegated UI usage	A consumer client can provide a frame for a provider UI and render the content	OSLC	http://open-services.net/bin/view/Main/OslcCoreSpecification?sortcol=table;table=up;up= - Delegated_User_Interface_Dialogs	

4.3.4 Advanced Core Capabilities

IOS	Capability	Description	Source Org	Source ref	Notes

ref.					
20	Service and User Authenticati on	Open standard for Authentication (OAUTH)	IETF	RFC5849 - The OAuth 1.0 Protocol	
			OSLC	http://open-services.net/resources/tutorials/oslc-primer/oauth/	Guideline

4.4 IOS Chapter – Domain Support: Requirement Management

IOS ref.	Capability	Description	Source Org	Source ref	Notes
21	Requirement	Primary Requirements Management artefact	OSLC	http://open-services.net/bin/view/Main/RmSpecificationV2	
22	Vocabulary for Resources	Requirements Management vocabulary	OSLC	http://open-services.net/bin/view/Main/RmSpecificationV2?sortcol=table;up= -RM_Resource_Definitions	
23	Linkage predicates	Requirements Management specification	OSLC	http://open-services.net/bin/view/Main/RmSpecificationV2?sortcol=table;up= -RM_Relationship_Properties	
24	Resource states	Requirement lifecycle states	OSLC	Not defined in OSLC V2.0 as not present in OSLC V2.0	
25	Issues	V2.0 Spec issues	OSLC	http://open-services.net/wiki/requirements-management/OSLC-Requirements-Management-version-2.0-issues/	

4.5 IOS Chapter – Domain Support: Architecture Management

IOS ref.	Capability	Description	Source Org	Source ref	Notes
26	Resource	Primary Architecture Management resource	OSLC	http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-Specification-Version-2.0/	
27	Vocabulary for Resources	Architecture Management vocabulary	OSLC	http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-Specification-Version-2.0/ -AM-Resource-Definitions	
28	Linktype	Architecture Management	OSLC	http://open-services.net/wiki/architecture-management/OSLC-Architecture-	

		specification		Management-Specification-Version-2.0/ - Resource_Link_Type_Resource_LTR	
29	Resource states	Resource lifecycle	OSLC	Not provided in IOS V1 as not present in OSLC V2.0	
30	Issues	OSLC V2.0 issues	OSLC	http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-v2.0-Specification-Issues/	

4.6 IOS Chapter – Domain Support: Asset Management

IOS ref.	Capability	Description	Source Org	Source ref	Notes
31	Asset	Primary Asset Management resource	OSLC	http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/ - Asset	
32	Artefact	Primary Asset Management resource	OSLC	http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/ - Artifact	
33	Artefact media	Primary Asset Management resource	OSLC	http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/ - Artifact_Media	
34	Vocabulary for Resources	Asset Management vocabulary	OSLC	http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification	See OSLC Asset: Start of additional properties
35	Linkage predicates	Asset Management specification	OSLC	http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification	See Relationship Properties
36	Resource states	Resource lifecycle	OSLC	Not provided in V1.0 as not present in OSLC V2.0	
37	Issues	OSLC V2.0 Issues	OSLC	http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-specification-issues/	

4.7 IOS Chapter – Domain Support: Change Request Management

IOS ref.	Capability	Description	Source Org	Source ref	Notes
38	Change Request	Primary Change Management artefact	OSLC	http://open-services.net/bin/view/Main/CmSpecificationV2?sortcol=table;up= - Resource_ChangeRequest	
39	Vocabulary for Resources	Change Management vocabulary	OSLC		

40	Linkage predicates	Change Management specification	OSLC	http://open-services.net/bin/view/Main/CmSpecificationV2?sortcol=table;up= - Resource_ChangeRequest	Relationship properties
41	Change Request states[1]	Change Management Specification	OSLC	http://open-services.net/bin/view/Main/CmSpecificationV2?sortcol=table;up= - State_Predicates	See also OSLC CM: Start of additional properties and State predicate properties
42	Issues	OSLC V2.0 issues	OSLC	http://open-services.net/wiki/change-management/Issues-2.0/	

Note: Neither IOS V1 nor OSLC V2.0 provide control over Artefacts through events e.g. like a POST. Artefact state change is achieved through normal use of PUT.

4.8 IOS Chapter – Domain Support: Quality Management

IOS ref.	Capability	Description	Source Org	Source ref	Notes
43	Test Plan	Primary Quality Management artefact	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2?sortcol=table;up= - Resource_TestPlan	
44	Test Case	Primary Quality Management artefact	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2?sortcol=table;up= - Resource_TestCase	
45	Test Script	Primary Quality Management artefact	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2?sortcol=table;up= - Resource_TestScript	
46	Test Execution Record	Primary Quality Management artefact	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2?sortcol=table;up= - Resource_TestExecutionRecord	
47	Test result	Primary Quality Management artefact	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2?sortcol=table;up= - Resource_TestResult	
48	Vocabulary for Resources	Quality Management vocabulary	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2	
49	Linkage predicates	Quality Management specification	OSLC	http://open-services.net/bin/view/Main/QmSpecificationV2	Relationship properties

50	Resource states	Quality Management Specification	OSLC	Not provided in IOS V1.0 as not present in OSLC V2.0	See status under additional properties
51	Issues	OSLC V2.0 issues	OSLC	http://open-services.net/bin/view/Main/QmSpecV2Issues	

5 CRYSTAL Engineering Methods to IOS Services

5.1 Engineering Method (EM) to IOS Services Mapping Methodology

For the CRYSTAL scenario based approach, it is important to identify where and which IOS functionality is needed to make the scenario interoperable. During analysis steps of the scenarios/use cases, the required building blocks (Bricks) are already identified and reusable "sub scenarios" (*Engineering Methods*, EMs) are defined as described in the overall *CRYSTAL Technical Management Process*.

The identification of the IOS needs is based on these EMs. In order to do this, the Use Case owners need to describe their EMs on a detail level, which allows assigning the necessary IOS functionality (IOS services) to the individual EM steps (e.g., the EM step "get a list of requirements" can be mapped to an "OSLC basic query service" provided by a requirement management brick).

For the Lifecycle Interoperability based on OSLC, many services are already defined by the OSLC standard and can be used as a first set to be assigned to EM steps. Missing services (Lifecycle Interoperability and others services) can be described in addition. If such a service is needed by multiple EMs, it can be a candidate for an IOS extension. If the service is not chosen to be a part of IOS, it can still be implemented as an individual solution for the EM. Detailed criteria for accepting services to be part of IOS still need to be defined, but the general applicability in multiple EMs and in multiple industrial domains as well as the chances for standardization will be among them.

This approach allows, and the Use Case owners are encouraged to, to define the IOS mapping in multiple iterations. In a first iteration, only those IOS services should be listed which are obvious or needed for a first implementation of the Use Case. With later iterations, more and more IOS services can be added.

Owners of the EM description and the IOS mapping are the Use Case owners, the IOS team has appointed IOS experts for each domain to help identifying the right IOS services. Also the brick owner of the bricks used for the EM should be involved in the IOS service identification process.

Training material for the IOS mapping has been provided and multiple webcasts have been conducted to train this mapping process.

Already during the first EM step to IOS service mapping exercises, this process has be shown to be very useful, not only to define the necessary services but also to clarify the engineering steps and make decisions regarding the implementation.

5.2 Engineering Method Steps – IOS service mapping table

In order to do the mapping of the EM steps and IOS service, the EM description template has been extended by the EM step to IOS service mapping table.

This table has the following columns:

IOS Domain: Identifies the IOS (engineering) domain where the service is selected from
Examples: OSLC-RM, OSLC-QM, OSLC-CM, FMI or other
The value "none" should be used if the EM step is completely handled within a brick and no interoperability with other bricks is required.

IOS Service: IOS service to be used for the EM step
Examples: create, read, update, basic-query, DUI-picker, export-FMU

IOS Service Detail (optional): Additional service parameters can be specified (e.g., properties to be used, special values required)

Brick Allocation (provider): Identifies the brick that need to provide the service (typically the data owner)

Brick Allocation (consumer): Identifies the brick that consumes the service (typically user interactions)

EM / EM Step: EM name as defined in the EM sheet of the same document
EM step number as assigned in the EM sheet of the same document

Comments: Any additional comment, e.g., for the implementer of the service, or remark why this IOS service was chosen in favour of another possible IOS service.

5.3 Engineering Method Steps – IOS service mapping table example

This is an (abbreviated) example of an EM step – IOS mapping table to illustrate how such a table can look like. It is a part of UC 401 of the Healthcare domain and describes IOS services used for the EM "Verify Requirements".

IOS Domain	IOS Service	IOS Service Detail (optional)	Brick Allocation provider	Brick Allocation consumer	EM / EM Step	Comments
					Verify Requirement	
none	none		IBM DOORS NG / B2.16	IBM DOORS NG / B2.16	1	no interaction with other bricks
OSLC-RM	Read		IBM DOORS NG / B2.17	PTC Windchill / B6.6	1a	
OSLC-RM	Update		IBM DOORS NG / B2.18	PTC Windchill / B6.6	1a	
OSLC-RM	Update		IBM DOORS NG / B2.19	PTC Windchill / B6.6	1a	
OSLC-RM	Read		IBM DOORS NG / B2.20	HP-QC / B4.12	2	
OSLC-RM	Basic Query		IBM DOORS NG / B2.21	HP-QC / B4.12	3	
OSLC-RM	Basic Query		IBM DOORS NG / B2.22	HP-QC / B4.12	3	
none	none		HP-QC / B4.12	HP-QC / B4.12	4-1	no interaction with other bricks
not_yet_defined	store_in_archive		document generation	document generation	5	to be defined in a later iteration
none	none		HP-QC / B4.12	HP-QC / B4.12	5	
OSLC-QM	update	http://open-services.net/ns/qm#TestPlan Update http://open-services.net/ns/qm#TestCase Update	HP-QC / B4.12	PTC Windchill / B6.6	6	Artifact types to be addressed in the IOS service listed as details
					<EM 2 name>	
					<EM step id>	
					<EM step id>	

Figure 10: Example of an EM – IOS Service Mapping table (excerpt)

6 Extension Mechanisms of IOS V1

6.1 CRYSTAL Extensions

As already described in section 2.4.3, the CRYSTAL IOS V1 has to be extended for supporting advanced lifecycle interoperability and in depth Systems Engineering activities as elicited by the CRYSTAL use cases. Pragmatically, there are three ways for envisaging extending the current version of the IOS for fulfilling integration needs and requirements not yet covered in V1, as detailed in the following:

1. Extensions of the IOS V1 chapters presented in section 4 can be proposed. Basically, it would simply consist in extending the IOS V1 resource vocabulary with additional linktypes and properties (e.g., with domain-specific, use-case-specific or brick-specific properties), or by proposing new resource types for these chapters.
2. New IOS chapters defining their own vocabularies (i.e., their own resource types with their sets of properties and linktypes) can be proposed for supporting advanced lifecycle interoperability scenarios specific to CRYSTAL needs, or new chapters can be simply reused/adopted from external sources (e.g., from other ARTEMIS projects such as MBAT, defining lifecycle vocabulary for supporting tight combination of testing and analysis methodologies, or from the OSLC community, e.g., defining vocabulary and services related to automation support, configuration management or reporting).
3. Finally, in order to support other interoperability topics related to in depth Safety-Critical Systems Engineering activities, other existing engineering standards can be adopted (see section 2.4.2), and will then consists of new IOS chapters as well. The content of such chapters in the subsequent version of the IOS deliverable is still to be established, but will most likely consist of (*i*) the definition of new lifecycle vocabulary for bridging the gap between these Engineering Standards and Lifecycle Interoperability, and of (*ii*) integration guidelines.

The inputs for such extensions will come from:

- External projects (such as MBAT) or communities (such as the OSLC community),
- From CRYSTAL domain ontology work packages (for the aerospace, automotive, healthcare and rail domains),
- From the CRYSTAL Technical Management Process that has been deployed in the project, and in particular from its *Engineering Method to IOS Services Mapping Methodology* (introduced in section 5). This methodology is driven by the CRYSTAL use case owners, and consists in allocating lifecycle related steps from the use cases onto CRYSTAL bricks and IOS Services (conducted with the brick providers from SP6 technical work packages and with the IOS experts from WP6.1). As a result of this process, clear gaps between the current IOS V1 and interoperability needs from the use cases will be identified, and will have to be fulfilled by CRYSTAL-specific IOS extensions.
- From the brick providers, e.g., for supporting specific methodologies implemented by their bricks.

The IOS extension candidates will be evaluated by WP6.1, and will be submitted to the CRYSTAL Technical Board for adoption before their publication in the subsequent versions of this deliverable.

6.2 Towards IOS Sustainability

In order to make the CRYSTAL Interoperability Specification a sustainable result, it is important to gather experience and to start communication with existing bodies regarding interoperability standardization (or de-

facto standardization). The CRYSTAL deliverable D601.031 (*Report on Standardisation Work - V1*[13]) reports on activities and achievements with regard to the standardization of the Interoperability Specification. Furthermore, within this deliverable, the alignments with other projects are documented and evaluated.

[13] https://projects.avl.com/11/0154/Data Exchange/006_Results/Deliverables_Submitted_to_JU/SP6/CRYSTAL_D_601_031_v1.0.pdf

7 Terms, Abbreviations and Definitions

Please add additional terms, abbreviations and definitions for your deliverable.

Table 1: Terms, Abbreviations and Definitions

Term	Description	Source
Brick	A tool conforming to the IOS	Crystal term
Capability	A function that achieves some useful measurable outcome	TOGAF DODAF
Engineering Method (EM)		Crystal specific
IT	Information Technology	
Reference Technology Platform (RTP)	The RTP (Reference Technology Platform) is a set of pre-integrated tools and services that can randomly be compiled and installed based on a respective business or engineering use case.	Crystal specific (from MBAT)
Resource	In general something made available on the web, in V1.0 this is using RDF	
Resource	A specific Resource (RDF) representation in the OSLC AM Spec	
Resource Description Framework	A means to model information resources in subject-predicate-object expressions, aka triples, specifically in V1.0 this is RDF/XML	W3C
Scenario	A path through or subset of a use-case	
Service	A mechanism to provide capabilities using a prescribed interface. Here we refer to IT based services.	
Tool	A software program which works with or on data to transform or product it	
Tool flow or Tool chain	Combination of tools and services which are used in scenarios, to fulfil the IT part a use case. The tool chain respects a specific work and data flow (engineering process, architecture).	Based on MBAT RTP V1.5
Tool Interoperability	The ability of two or more tools to exchange information and to use the information that has been exchange	Based on IEEE610
Use case	A description of the steps through an expected functionality of a system to reach a business goal.	Based on MBAT RTP V1.5

8 References

OASIS OSLC http://www.oasis-oslc.org/node/2	
OASIS OSLC Community site http://open-services.net/	
CRYSTAL Deliverable D601.031 - Report on Standardisation Work - V1	https://projects.avl.com/11/0154/Data Exchange/006_Results/Deliverables_Submitted_to_JU/SP6/CRYSTAL_D_601_031_v1.0.pdf

9 Annex

Annex I: Example of Concrete Scenarios

Table 2: Examples of Concrete Scenarios related to "Lifecycle Interoperability" and "other Interoperability Topics"

Lifecycle Interoperability Scenarios	Other Interoperability Topics related to Lifecycle Interoperability Scenarios	Other Interoperability Topics NOT related to Lifecycle Interoperability Scenarios
Traceability between Artefact Elements to requirements and test/analysis cases		
Querying lifecycle artefacts (e.g., Requirements) from multiple repositories		
Browsing the changes history on artefacts from multiple tools and repositories		
Displaying the impact of a requirement update on a set of design artefacts and simulation tasks		
	FMI for co-simulation with a mapping, e.g., from FMI simulation inputs and results onto IOS-based lifecycle artefacts traced back to requirements and design artefacts	FMI for co-simulation and model exchange
	Relating ECU measurement and calibration data from ASAM Standards to other Lifecycle Artefacts (e.g. Requirements)	Gathering and transferring of ECU measurement and calibration data based on the ASAM standard
	Traceability between Requirements and EAST-ADL Models from multiple repositories	Traceability between Requirements and Models based on EAST-ADL repository
Creating a list of requirements using selection criteria from different repositories		
		Exchange a list of requirements in ReqIF Format via offline collaboration to an external project partner
		Relating artefacts within one repository according to a proprietary format
		Creating a list of requirements based on attributes in one requirement proprietary repository
Relating artefacts of the same		

type (e.g. change request) between different instances of the same tool		
		Working toolset with established interoperability within one client (e.g., with Modelbus, Eclipse or vendor proprietary solutions)
Relating artefacts stored in a central repository (e.g., Requirement) with standard artefacts in a vendor specific toolset		
Change Impact Analysis on multiple artefact types (e.g., change on a requirement impacting on design models, test cases, formal analysis results, etc.)		
	Semantic-preserving model transformations with traceability between input and output artefacts	
Merging on a dashboard attributes from artefacts manipulated at different engineering phases which are of relevance for a particular tool chain stakeholder (e.g., analyst, project manager, Testing/Analysis expert)		