Open Services for Lifecycle Collaboration (OSLC) Linking profiles are non-normative supplement to OSLC specifications that specify additional constraints on OSLC conformance clauses and vocabulary terms to reduce variability and facilitate interoperability between OSLC enabled applications when linking across applications, domains, and heterogenious domain providers.

Introduction

Linking profiles are non-normative supplement to OSLC specifications, aiming to ensure interoperability between OSLC enabled applications when linking across applications, domains, and heterogenious domain providers. It is important to note that OSLC interoperability is not automatically implied by OSLC specifications, as the specifications allow for high degree of conformance variability.

Linking use cases result in establishing OSLC links across OSLC resources in different applications, possibly within the same OSLC domain or across domains. For example, a requirements provider from one vendor establishes traceability links to a requirements provider from another vendor. A cross domain example would be a PLM BOM application establishing traceability links to a requirements application by a different vendor. The cross vendors emphasis is because this is where there are interoperability challenges. Linking use cases may vary in capability as described later in this document, but all linking use cases require establishing connections between the resources in the consumer and provider applications, ability to select a resource with the provider to link with a resource of the consumer, and storage of the link in one of the applications. From that point onwards there is a variability across the use cases related to bi-directional navigability (are both applications aware of the link) and also behaviors that depend if the application supports configuration management or not.

The introduction of the linking profile helps users qualify whether two OSLC applications meet a compatibility level to perform a set of OSLC linking use cases. The linking profile also helps software vendors to know how to build/consume OSLC linking related services that ensure interoperability.

The linking profiles specify 4 (3+1 future) levels of conformance. Each aims at certain implementation level and set of linking use cases, and can also be considered as “OSLC maturity levels”. The profiles are based on common practice. Since IBM ELM has the longest and broadest OSLC integration adoptions of OSLC integrations, IBM ELM is used as a baseline for spec variability choices, so the profiles will ensure ELM linking interoperability. However, the profiles are meant provide an interoperability baseline to any other vendor and future interoperability related or unrelated to IBM OSLC applications.

Motivation

The fundamental set of interoperability use cases in the context of OSLC linked data architecture describe cross linking across lifecycle resources and domain providers. Cross linking forms the foundation for “digital continuity” and establishment of “digital threads”. Using linking profiles guarantees a set of use cases across providers implemented by different software vendors.

When originally developed, the OSLC specifications (Core, RM, CM, QM, AM, GCM) were purposefully kept general avoiding over-prescription. The intent was to allow a considerable degree of implementation variability to speed vendor adoption as well as let adopting vendors identify areas (though their adoption) that needed more prescription.by. This is reflected in the minimal set of mandatory (“MUST”) services , and rest that are only recommended (“SHOULD”) or optional (“MAY”).

The goal of interoperability across different vendor OSLC specification implementations requires peer to peer testing and adjustments to satisfy certain common interoperability use cases. Also, implementors complain that practically implementing interoperable OSLC providers or consumers is very time consuming and requires lots of discovery, trial and error. Additionally, when vendors label their applications as “OSLC compliant”, the possible implementation variability makes that label definition very broad becoming difficult for end users to know for certain what optional capabilities are supported that impact interoperability.

In addition to the variability caused by the mandatory/optional requirements, implementors also struggle to follow the specification without detailed explanations or examples. Unfortunately, while some guides exist, they are not always known about or found, requiring web searches. In some cases, guides simply do not exist. While OSLC does define discovery services, OSLC discovery does not provide a machine readable, standard way to determine which optional OSLC features a server implements. This has to be manually documented by server providers, and may not be easily reflected from the server’s APIs.

Solution scope

The linking profile specifies profiles based on maturity and use cases to reduce interoperability across clients and servers complying to a specific profile, and provides guidance with examples to eliminate ambiguities and confusion related to the implementation of the specifications. A key aspect of reducing spec variabilities is converting “SHOULD” and “MAY” clauses in the spec to “MUST” in certain profiles to ensure those features are implemented by the conforming servers. Guidance may be references existing articles and/or include supplementary information in the profile.

To enable server implementation flexibility with more predicable variability, the linking profiles accommodate various degrees of maturity and needs of different OSLC server implementations. The profile conformance levels are based on two key characteristics related to providers/consumers:

1. The need to maintain bi-directional navigability or not
2. Usage of configuration management by the provider or not

Linking Profile Conformance Levels

The integration challenges described above result from the amount of server implementation variability allowed by the OSLC specification, through the MAY and SHOULD conformance clauses. To simplify and improve interoperability across OSLC applications, while maintaining levels of variability that have been shown to be useful and effective in practice, the Linking Profile introduces the following conformance levels. Each level defines which capabilities are necessary to support the integration services specified for that level. This ensures that tools that advertise conformance to a Linking Profile level will have a predecitable, minimal viable set of capabilities that other clients and servers can rely on.

Basic Linking Profile

The **Basic**: Focuses on establishing links to a provider, without bi-directional navigation. The main objective of this profile is to establish proper connection with a provider by a consumer and obtaining a resource selection UI. The challenges related to the basic profile are around establishing OSLC connection with a provider with proper authentication and discovery of services.

For the Basic RM/AM/QM/CM Linking profile 3,0, a user needs to be able to make a link from inside the web interface of Tool A to an RM/AM/QM/CM resource inside Tool B. The user does not have a link to the resource in advance. Instead, the user shall be able to pick a resource when establishing a link. After selecting the resource, Tool A shall store the link to the resource and display the link title and icon.

Bi-directional Linking Profile

The **Bi-directional**: profile extends the capabilities of the Basic profile, with the ability to have by-directional navigation across the two applications, and the ability to preview linked resources in different applications.

For the Bidirectional RM/AM/QM/CM Linking profile 3.0, a user needs to be able to make a link from inside the web interface of Tool A to an RM/AM/QM/CM resource inside Tool B. The user does not have a link to the resource in advance. Instead, the user shall be able to pick a resource when establishing a link. After selecting the resource, the link is either stored in Tool A and/or in Tool B depending on the domain model requirements or how the tools support primary and secondary links. When a user navigates to one of the linked resources in a tool that does not store the link, the backlink or secondary link is shown.

Additionally, when the user of the Tool A hovers over a link to the resource, a preview dialog is shown next to the link.

Configuration Management Linking Profile

The **Config management Linking**: extends the Bi-directional linking profile with support for linking across configuration management enabled OSLC providers. This profile clarifies where a physical link is actually stored, and how the non-storing application can discover incoming links. The issues related to this profile are around link storage, discovery, and OSLC query.

For the Configuration-aware RM/AM/QM/CM Linking profile 3.0, the Bidirectional RM/AM/QM/CM Linking profile 3.0 applies. Additionally, both tools are configured with OSLC Configuration Management enabled.

Additionally, when the user of the Tool A hovers over a link to the resource, the preview dialog is shown next to the link and displays correct information for the configuration context chosen.

Link Discovery Linking Profile

The **Link Discovery**: profile extends the Configuration Management Linking Profile with a link discovery service for accessing incoming links.

For the Link Discovery RM/AM/QM/CM Linking profile 3.0, the Configuration-aware RM/AM/QM/CM Linking profile 3.0 applies. Additionally, Additionally, all tools contribute outgoing links to a Link Discovery Management server, and get all incoming links from a Link Discovery Management server.

Solution Capability Matrix

The following table summarizes the OSLC capabilities for each of the profiles.

Capability	Basic	Bi-Directional	Config	Link Discovery
Root Services document	MUST	MUST	MUST	MUST
OIDC Authentication	MUST	MUST	MUST	MUST
OAuth 2.0 Authorization	MUST	MUST	MUST	MUST
CSP for friends	MUST	MUST	MUST	MUST
CORS for friends	MUST	MUST	MUST	MUST
Selection Dialogs	MUST	MUST	MUST	MUST
Preview Dialogs		MUST	MUST	MUST
Link Ownership			MUST	MUST
PUT on Resources		MUST	MUST	MUST
OSLC Query			MUST	MUST
Config Management			MUST	MUST
OSLC Link Discovery Service				MUST

The sections below provide details for each profile capability including:

1. The applicable profiles
2. The applicable OSLC conformance clauses
3. How those conformance clauses are tightened in the profile
4. Any additional ResourceShape constrints
5. The intended purpose of the additional constrats
6. Any additional conformance clauses
7. Examples

In the sections below, OSLC Server refers to an OSLC server that conforms to or supports the required capabilities of one or more of the Linking Profile conformance levels.

An OSLC Server MUST support the RDF/XML resource serialization format, and SHOULD support as many serialization formats as possible through content negotiation.

Root Services

Capability	Basic	Bi-Directional	Config	Full

Capability	Basic	Bi-Directional	Config	Full
Root Services document	MUST	MUST	MUST	MUST

OSLC Core, section 4.2 Well-known URI Bootstrapping recommends the use of a rootservices document to list the service providers and tracked resource sets an OSLC server provides. The format of the rootservices document used by IBM ELM jazz.net products is described in Root Services Specification. An OSLC Server MUST support rootservices to provide a standard way of “discovering OSLC discovery”.

Authentication

Capability	Basic	Bi-Directional	Config	Link Discovery
OIDC Authentication	MUST	MUST	MUST	MUST
OAuth 2.0 Authorization	MUST	MUST	MUST	MUST

OSLC Core Version 3.0. Part 1: Overview recommends specific approaches for authentication and secure communication between clients and servers, as well as between servers. These align with the industry best practices for RESTful APIs and linked data at the time the OSLC Core specification was developed.

However, the OSLC Core authentication recommendations have proven to be insufficiently specific to facilitiate secure integration scenarios that required authentication and authorization. As a result, a signifiant barrier developing and using OSLC integrations has been establishing authentication and authorization. Since OSLC Core was developed, the IT industry has focused a lot more concern and effort on security. The following specific recommendation are for the linking profiles tighten the OSLC Core recommendations based on current industry best practices.

Client-to-Server Authentication

OSLC servers MUST support OpenID Connect for client to server authentication and authorization.

OSLC suggests using **standard HTTP-based authentication mechanisms** to ensure secure access to RESTful services. OSLC Clients should be prepared to handle authentication challenges from the recommended methods including:

OAuth 2.0 (Preferred)

- **Recommended Flow: Authorization Code Grant** (for web apps) or **Resource Owner Password Credentials** (for trusted clients).
- **Use Case:**
  - Third-party applications accessing OSLC services on behalf of a user.
  - Fine-grained access control via scopes (e.g., `read`, `write`, `update`).
- **Implementation:**
  - Clients obtain an access token from an OAuth 2.0 provider (e.g., Keycloak, Okta).
  - Token is sent via the `Authorization: Bearer <token>` header.

OpenID Connect (OIDC)

- Extends OAuth 2.0 for authentication or identity verification.
- Provides **user authentication** + **OAuth 2.0 authorization**.
- Used in enterprise SSO (Single Sign-On) scenarios.

HTTP Basic Authentication (Legacy)

- **Not recommended for production** unless over HTTPS.

- Credentials sent as Authorization: Basic <base64(user:password)>.
- OSLC servers may support this for backward compatibility.

JEE Forms Authentication (Legacy)

- **Not recommended for production** unless over HTTPS.
- Credentials sent as j_username and j_password parameters to a /j_security_check authentication URL in response to an authentication challenge in a x-com-ibm-team-repository-web-auth-msg=authrequired header
- OSLC servers may support this for backward compatibility.

Mutual TLS (mTLS)

- Used in high-security environments (e.g., IoT, industrial systems).
- Both client and server authenticate via X.509 certificates.

Server-to-Server Authentication

OSLC servers MUST support OAuth 2.0 for server to server authorization.

For automated interactions (e.g., CI/CD pipelines, federated OSLC servers), OSLC recommends:

OAuth 2.0 Client Credentials Flow

- **Use Case:** Machine-to-machine (M2M) communication.
- **How it Works:**
  1. Servers register as OAuth clients and obtain a client_id + client_secret.
  2. They request an access token using these credentials.
  3. Token is used in API requests (Authorization: Bearer <token>).

OAuth 1.0a (Legacy Approach)

1. Server A pre-registers a shared secret with OSLC Server B.
2. For each API call, Server A must:

- Generate a signature using the secret + request params.
- Include the signature in headers (oauth_signature).

3. Server B validates the signature before responding.
4. Often required for IBM ELM OSLC integration

Mutual TLS (mTLS)

- Servers authenticate each other via **X.509 certificates**.
- Common in **private cloud/on-premise OSLC deployments**.

API Keys (Less Secure)

- Some OSLC implementations may use API keys (sent in headers or query params).
- **Not recommended** for sensitive data (lacks fine-grained revocation).

Additional Security Recommendations

- **Always use HTTPS (TLS 1.2+)** to encrypt all communications.
- **Token expiration & refresh:** Short-lived access tokens with refresh tokens.
- **CORS & CSRF protection:** OSLC servers should restrict cross-origin requests appropriately.
- **Rate limiting:** Prevent abuse of OSLC APIs.

CSP for friends

Capability	Basic	Bi-Directional	Config	Full
CSP for friends	MUST	MUST	MUST	MUST

Content Security Policy (CSP) is a web security standard that helps protect against cross-site scripting (XSS) and other code injection attacks. It works by giving website developers control over which resources (scripts, images, etc.) the browser is allowed to load and execute for a given page. By whitelisting trusted sources, CSP helps prevent browsers from loading malicious or untrusted content, even if a user has been tricked into visiting a compromised website.

An OSLC Server MUST support the CSP Content-Security-Policy HTTP response header so that modern browsers can use it to enhance the security of the HTTP entity response resource. The Content-Security-Policy header allows servers to restrict which resources (such as JavaScript, CSS, Images, etc.) can be loaded, and the URLs that they can be loaded from.

For example, this response header:

```
Content-Security-Policy: default-src 'self'; img-src 'self' cdn.example.com;
```

tells the receiving browser to only allow resource URLs frorm the same origin (domain and scheme), and to allow access to images from the same origin and the cdn.example.com domain.

CORS for friends

CORS (Cross-Origin Resource Sharing) is a security mechanism that allows or restricts web applications running in a browser to make requests to a different domain (origin) than the one they originated from. This is commonly used in OSLC resource preview and delegated selection and creation dialogs where an OSLC client running in a HTTP browser needs to access resources from multiple servers. This is required to allow the creation and navigation of links between resources in different OSLC servers.

By default, browsers block web pages from making requests to a different domain (origin) for security. CORS identifies the request origin as a combination of the protocol (http/https), host domain (example.com) and port (:8080).

CORS provide a means for clients and servers to control the Same-Origin Policy to allow controlled cross-origin requests. When a browser makes a cross-origin request, it first sends a preflight request (OPTIONS) to check if the server allows it. The server responds with CORS headers indicating permitted origins, methods, and headers. This ensures that only cross-origin requests that have been explicitly configured are permitted.

An OSLC Server must support CORS as described in Fetch This ensures that security administrators can have sufficient control of cross-origin requests to avoid security issues.

Selection Dialogs

Capability	Basic	Bi-Directional	Config	Full
Selection Dialog	MUST	MUST	MUST	MUST

An OSLC Server MUST support OSLC Discovery and MUST support OSLC Selection Delegated Dialogs. An OSLC Server SHOULD support OSLC Creation Delegated dialogs. These requirements are necessary to support creating links between OSLC resources. An OSLC Client discovers the service by looking for oslc:selectionDialog property and making a GET request on the service resource.

Link Ownership

Capability	Basic	Bi-Directional	Config	Full

Capability	Basic	Bi-Directional	Config	Full
Link Ownership			MUST	MUST

In bi-dirctional linking scenarios both OSLC participants are aware of links across their owned resources and enable link visibility and navigation on each side. In the Bi-Directional profile, OSLC Servers can support bi-directional links, links in the form of (source, primary-predicate, target) and {target, secondary-predicate, source) however they wish. This could be done for example by storing backlinks, or using queries to get links. However, storing a link and a backlink at both sides does not follow OSLC Link Guidance as it is essentially replication of data. This may result in inconsistencies as links are updated or deleted, since maintaining consistency requires synchronization across the providers on any update. Therefore the recommended practice is to store links on one of the participants, and use link discovery by the other participant. Therefore there needs to be an agreed convention on which side should store the link.

Many OSLC links have an incoming and outgoing sides, determined by the direction of the link. Usually one side will have an active predicate name, for example, “implements” and the other side will have a passive predicate name, such as for example “implemented by”. The active side is often considered the outgoing side, and the passive the incoming side. The convention is that the link is typically stored with the resource on the outgoing side, i.e., the resource with the active predicate. The incoming side would discover the links with one of the discovery methods discussed in the following sections.

Note the creation of the link can be initiated from the OSLC server at either end of the link. In case that the link is initiated by the incoming side provider, it needs to store it with the resource on the outgoing side provider. This is discussed in the PUT on Resources section. In addition, for historical reasons, certain OSLC providers, such as IBM ELM, may have some variations of behaviors between configuration enabled and non-configuration enabled modes, where in non-configuration modes there may still be a usage of backlink storage. In addition, providers do not support OSLC versioned resources such as CM (change management servers) would have preference for link storage over configuration enabled providers, to prevent baselined links to refer to mutable entities.

Based on these principles and conventions, the Linking Profile establishes specific ownership for OSLC link properties. OSLC Servers conforming to the Bi-Directional linking pofile should store primary predicate links on the source/owner domain side as descirbed in the following table. OSLC Servers conforming to the Config or Full linking pofiles MUST store primary predicate links on the source/owner domain side.

Source/Owner domain	Primary predicate	Target domain	Secondary predicate
RM	oslc_rm:constraints	RM	oslc_rm:constrainedBy
RM	oslc_rm:decomposes	RM	oslc_rm:decomposedBy
RM	oslc_rm:elaborates	RM	oslc_rm:elaboratedBy
RM	oslc_rm:satisfies	RM	oslc_rm:satisfiedBy
RM	oslc_rm:specifies	RM	oslc_rm:specifiedBy
RM	oslc_rm:uses	RM	-unspecified-
QM	oslc_qm:validatesRequirement	RM	oslc_rm:validatedBy
QM	oslc_qm:validatesRequirementCollection	RM	oslc_rm:validatedBy
CM	oslc_cm:implementsRequirement	RM	oslc_rm:implementedBy
CM	oslc_cm:tracksRequirement	RM	oslc_rm:trackedBy
CM	oslc_cm:affectsRequirement	RM	oslc_rm:affectedBy
CM	oslc_cm:testedByTestCase	QM	oslc_qm:testsChangeRequest
CM	oslc_cm:relatedTestScript	QM	oslc_qm:relatedChangeRequest
CM	oslc_cm:relatedTestCase	QM	oslc_qm:relatedChangeRequest

Source/Owner domain	Primary predicate	Target domain	Secondary predicate
CM	oslc_cm:relatedTestPlan	QM	oslc_qm:relatedChangeRequest
CM	oslc_cm:relatedTestExecutionRecord	QM	oslc_qm:relatedChangeRequest
CM	oslc_cm:blocksTestExecutionRecord	QM	oslc_qm:blockedByChangeRequest
CM	oslc_cm:affectsTestResult	QM	oslc_qm:affectedByChangeRequest
CM	oslc_cm:affectedByDefect	CM	oslc_cm:affectsPlanItem
CM	oslc_cm:tracksChangeSet	oslc_config:ChangeSet	-unspecified-
CM	oslc_cm:relatedChangeRequest	CM	-unspecified-
AM	jazz_am:derives	oslc:Any	-unspecified-
AM	jazz_am:elaborates	oslc:Any	-unspecified-
AM	jazz_am:external	oslc:Any	-unspecified-
AM	jazz_am:refine	oslc:Any	-unspecified-
AM	jazz_am:satisfy	oslc:Any	-unspecified-
AM	jazz_am:trace	oslc:Any	-unspecified-

All links are directional from source to target e.g., Requirement testedBy TestCase, TestCase validates Requirement. OSLC does not define inverse links, however, links are often paired using active/passive names as in implements/implementedBy, validates/testedBy, etc.

There is some interest in extending OSLC ResourceShapes to formalize these active/passive relationships between link types so their property URLs and titles can be discovered in a standard way.

Links have to be created between existing source and target resources (the targets either already exist and are selected, or are created before the link is created).

Links are only stored in the source resource on one side (usually the active side) of related link types. The “inverse” or incoming link is typically displayed, but is obtained through a query rather than being stored. This avoids storing redundant data that can lead to update anomalies.

By convention, links are stored on the “downstream” side, i.e., the resource that is created later in the lifecycle. This ensures that both the source and target of the link are likely to exist when the link is created. For example, in Requirements-Driven-Development, requirements are created first, and then later in the lifecycle, test cases are created that validate the requirements. When the test case is created, the requirement will most likely already exist, so the link is stored in the downstream test case source as TestCase validates Requirement.

Note however that if an organization used Test-Driven-Development, test cases are often created first, representing an executable expression of a requirement that might be identified and refined later. So the convention to store RM/QM links on the QM side is not always ideal, but does provide consistency that improves and simplifies overall tool integration.

Storing related links on only one side is also necessary when using configuration management since it must be possible to create a link from a source to a target resource without changing the target resource version (it could be in a baseline).

For example, the requirements for a project may have already been created and baselined before test cases are created. Then when the test cases are created, they can be linked to the requirement they validate without changing the requirement.

Similarly, a versioned resource cannot store links to unversioned resources because it would be impossible to create an immutable baseline of such a resource since the target unverioned resource could changed or deleted, indirectly modifying an immutable source resource.

PUT on Resources

An OSLC Server MUST support PUT on resources to allow for the creation and storage of links. When an OSLC Server that does not own the link initiates link creation using the selection dialog of a server that does own the link, then the initiating server needs to GET on the selected resoures, add an assertion for the link, using the secondary predicate, and then PUT to the OSLC Server owning the link to save the link. Primary and secondary predicates are listed in the table above.

Preview Dialogs

Capability	Basic	Bi-Directional	Config	Full
Preview Dialogs		MUST	MUST	MUST

An OSLC Server MUST support OSLC Resource Preview to allow applications to preview resources across links.

OSLC Query

Capability	Basic	Bi-Directional	Config	Full
OSLC Query			MUST	MUST

An OSLC Server MUST support OSLC Query, minimally the oslc.select with nested properties, and oslc.where clauses. This allows OSLC Clients and Servers to have a standard means of querying links and accessing information about linked resources.

Configuration Management

Capability	Basic	Bi-Directional	Config	Full
Config Management			MUST	MUST

An OSLC Server conforming to the Config management Linking or Full linking profiles MUST support configuration management. The Configuration Management profile, as an extension to the Bi-Directional linking profile, supports creating and navigating versioned links in both directions. The Configuration Management profile does not specify how the server access incoming links. OSLC Servers could use OSLC query for accessing incoming links, and the OSLC Query capabilities specified in this document are intended to be necessary and sufficient for this purpose.

OSLC Link Discovery Management

Capability	Basic	Bi-Directional	Config	Full
OSLC Link Discovery Management				MUST

The Full linking profile introduces a standard means for accessing incoming and optionally, outgoing links. This extends the Config linking profile to provide predictable, high performance, reliable versioned link creation, update and navigation in a distributed environment consisting of federated servers supporting different lifecycle domains. OSLC Servers supporting the Full linking profile MUST support the OSLC Link Discovery Management Server version 1.0.

It is recognized that the link ownership rules encourage specific workflows. For example, the fact that QM owns the oslc_qm:validatesRequirement link implies that users using a requirements-driven methodology would likely create requirements before they create test cases to validate them, thus ensuring the requirements exist when the user cratest the oslc_qm:validatesRequirement link. However, test-driven development methodologies might create the test cases first, perhaps as an executable representation of a requirment that can be then elaborated with additional oslc_rm:validatedBy requirements.

But implementing the linking profiles does not restrict severs to follow specific, overly rigid methodologies: 1. Selection and Creation dialogs can be used to create primary or secondary links from either side 2. The PUT operations allows a server that doesn't own the link to add the link to the resource owned by another server and ask that server to update its resource. 3. The link ownership table specifies what server owns what links, and the primary and secondary link predicates so that server implementors know how to create links regardless of which server owns them. 4. The Link Discovery Management provides an efficient means for accessing incoming links in a standard way.

So link ownership, or where links are stored, can be transparent to users, while being predicatable for different server implementations to ensure consistent integrations of versioned links across OSLC domains and servers.

Users should typically not be aware of where links are stored, and should be able to create links from either direction in a transparent way. For primary links the server owns, the server can easily create, store, access, navigate and display these links. However, for secondary link, the information is stored in a different server, and to display them, a server would need to access incoming links, the links for which its resource is the source of the link.

Servers can use OSLC query as a standard means of requesting incoming links. However, this does not scale well when a server has to access incoming links from many other servers.

OSLC defines a Link Discovery Management specification as a standard means of accessinig incoming versioned or unversioned links. Bi-Directional, Config and Full link profiles must support the LDM specification.

Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Project Governing Board:

James Amsden, MID
Ernest Mah, IBM

Techical Steering Committee:

James Amsden, MID
Andrii Berezovskyi, KTH
Jad El-khoury, Lynxwork

Additional Participants:

Eran Gery, Sodius-Willtert
Andrii Berezovskyi, KTH
Jim Gammon, Raytheon

