



SOLA Solutions for Open Land Administration

Software Architecture Document

for the SOLA Development Snapshot



Revision History

Date	Version	Description	Author
10 Feb 2011	0.1	Initial Draft	Andrew McDowell
15 Feb 2011	0.2	Second Draft	Andrew McDowell
16 Feb 2011	0.3	Internal Release	Andrew McDowell
6 Mar 2011	1.0	Release	Andrew McDowell
15 Aug 2011	1.1	Revised for the Development Snapshot	Andrew McDowell



Table of Contents

1.	Introduction	5
1.1	Scope	5
1.2	Intended Audience	5
1.3	Architectural Representation	5
1.4	References	5
1.5	Acknowledgements	6
2.	Overview	7
2.1	Presentation Layer	8
2.2	Services Layer	8
2.3	Data Layer	8
2.4	External Systems	8
3.	Architectural Drivers	9
3.1	Requirements	9
3.2	Constraints	15
3.3	Principles	15
4.	Architectural Mechanisms	18
4.1	Analysis Mechanisms	18
4.2	Analysis Mechanism Mappings	19
5.	Use Case View	21
5.1	Architecturally Significant Use Cases	21
5.2	Use-Case Realisations	23
6.	Logical View	34
6.1	Overview	34
6.2	Presentation Layer	35
6.3	SOLA Desktop	35
6.4	SOLA GIS and GeoTools UI	38
6.5	Services Layer	40
6.6	Boundary	41
6.7	EJBs	43
6.8	Application EJB	44
6.9	Services Common	45
6.10	Common	47
6.11	Data Layer	47
6.12	External Systems	48
7.	Deployment View	49
7.1	Deployment Models	49
7.2	Deployment Artefacts	49
7.3	SOLA Test	50
7.4	SOLA Development Snapshot	51
7.5	SOLA Developers Package	52
8.	Data View	53
8.1	SOLA Data Model	53
9.	Size and Performance	54



9.1	Use Case Points	54
9.2	Database Size	54
9.3	Code Metrics	54
9.4	Performance	54
10.	Security	55
10.1	SOLA Security Model	55
10.2	Authentication	55
10.3	Authorization	56
10.4	Confidentiality and Integrity	56
10.5	Auditing and Logging	57
10.6	Exception Management	57
10.7	Map Viewer Navigation	58
11.	Appendix 1 – Definitions, Acronyms and Abbreviations	59
12.	Appendix 2 – UML Notation	62
12.1	Component Model Notation	62
12.2	Use Case Model Notation	62
12.3	Logical View Notation	63
12.4	Deployment Model Notation	64
13.	Appendix 3 – Architectural Risks	66
14.	Appendix 4 – Additional Information	68
14.1	Workflow Engine Evaluation	68
14.2	Enterprise Service Bus	70



Software Architecture Document

1. Introduction

This document provides an architectural description of the Solutions for Open Land Administration (SOLA) system using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the architectural decisions that have been made and elaborates on aspects of the system that are considered to be architecturally significant.

The views include use case view, logical view, deployment view and data view. Also described are the drivers that have shaped the architecture of the system, the architectural mechanisms applying to SOLA and its size, performance and security characteristics.

1.1 Scope

This Software Architecture Document (SAD) represents the “as-is” architecture of the SOLA Development Snapshot. The Development Snapshot is a branch of the SOLA source code and new areas currently under development in the main branch are also discussed and represented in this document with light fill.

The SOLA project is using the Agile Scrum development methodology. This document will evolve during project Sprints and additional detail (generally marked as “To be completed”) will be included as appropriate to reflect decisions and outputs arising from design and implementation activities.

1.2 Intended Audience

This is a technical document intended for developer and related technical resources.

1.3 Architectural Representation

The Unified Modelling Language (UML) is an industry standard software system modelling notation administered by the Object Management Group.

This document describes the architecture of SOLA using standard UML diagrams, including use case, component, package, and deployment diagrams. Readers of this document need to be familiar with the UML notation. For further information on the UML see <http://www.uml.org/>. Note that a brief description of the UML notation used in this document is provided in Appendix 2 – UML Notation to assist the reader.

1.4 References

Document Name	Date	Author
FLOSS SOLA Data Dictionary v1.0	26 Jul 2011	SOLA Development Team
Statement of Requirements – Initial Solutions for Open Land Administration (SOLA) Software v1.1	14 Jan 2011	Neil Pullar
Technology Options for SOLA	Jan 2011	A McDowell
SOLA LADM v0.3 Enterprise Architect Model	8 Feb 2011	Neil Pullar
FAO SOLA Way of Working Document v1.1	10 Feb 2011	Alexander Lokhman and Imed Hammouda
The Java EE 6 Tutorial (http://download.oracle.com/javaee/6/tutorial/doc/docinfo.html)	Nov 2010	Oracle Corporation



Document Name	Date	Author
PostgreSQL Site (http://www.postgresql.org/)	2011	PostgreSQL User Community
Metro Users Guide (http://metro.java.net/guide/)	01 Feb 2011	Metro User Community
Real World Java Patterns – Rethinking best practices	June 2009	Adam Bien
ObjectDB JPA Reference (http://www.objectdb.com/api/java/jpa)		ObjectDB
Enterprise Architecture Patterns (http://martinfowler.com/eaDev/)	July 2006	Martin Fowler
Netbeans Documentation, Training and Support (http://netbeans.org/kb/index.html)	2011	Oracle Corporation
Sun Developer Network Java Internationalization (http://java.sun.com/javase/technologies/core/basic/intl/)	2010	Oracle Corporation
GeoTools documentation (http://docs.geotools.org/)	2011	GeoTools

1.5 Acknowledgements

The following people have provided information or feedback that has contributed to the content of this document

- Neil Pullar
- Geoff Hay
- Elton Manoku
- Alexander Solovov
- Rafiq DounNaah
- Matthew Bofo
- Paola Rizzo



2. Overview

The architecture of SOLA has been designed using the layered architectural pattern. A two dimensional layering approach has been used to structure the software firstly by responsibility and secondly for reuse. The following Component Model diagram illustrates the layers of SOLA and the main components within each layer along with their key dependencies.

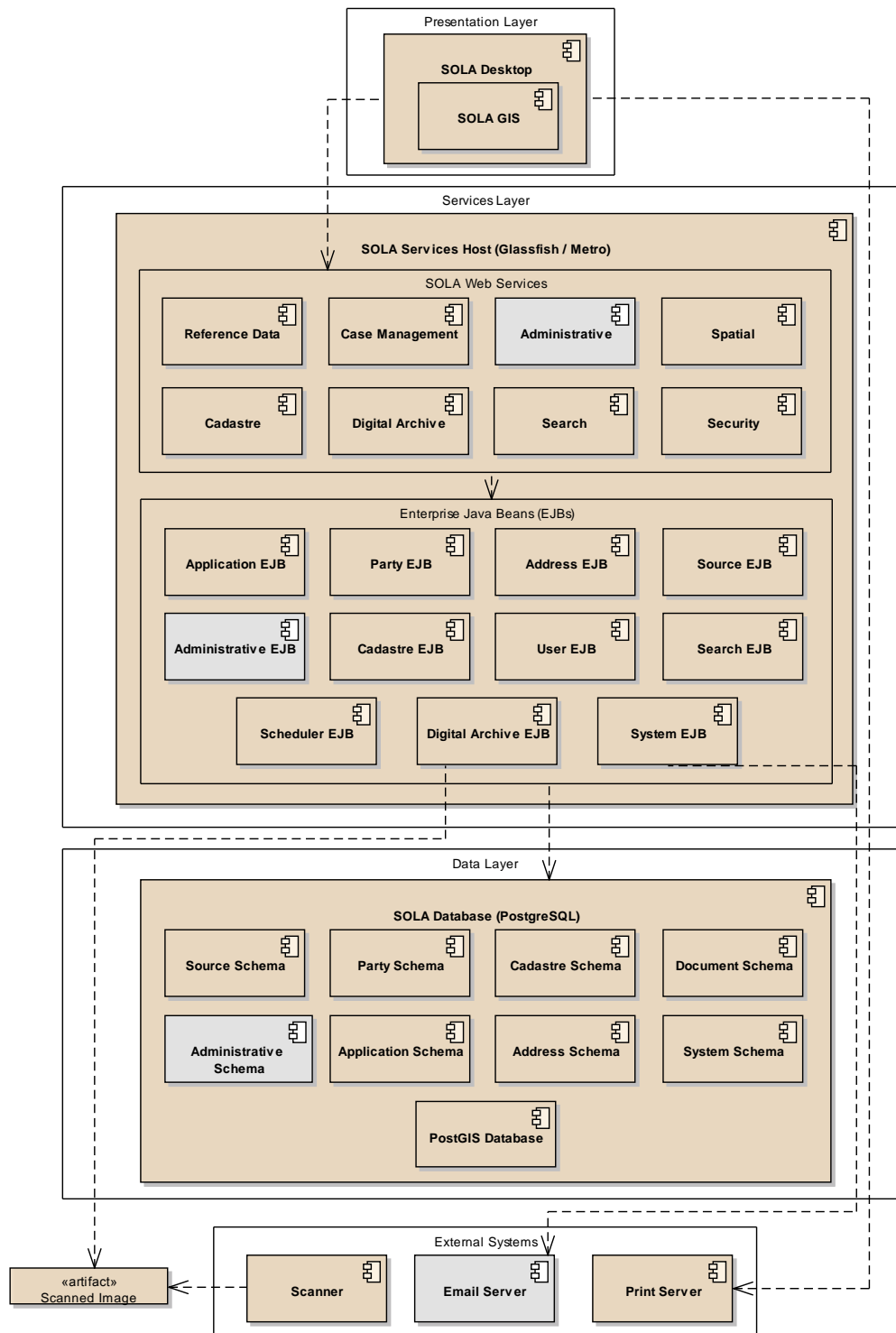


Figure 1 - SOLA Overview



2.1 Presentation Layer

The SOLA Desktop is a Java Swing desktop application delivered to end users via Java Web Start technology to ensure minimal deployment and configuration overhead. The Development Snapshot version of the SOLA Desktop supports basic front office case management tasks including application lodgement, digital document preview and attachment, application search, application assignment and a custom spatial map viewer (SOLA GIS). The SOLA Desktop provides translations for both the English and Italian languages and can be configured to support additional languages.

The SOLA Desktop will continue to be extended to support the Generic Land Administration Process¹. Given each land administration agency will have different workflow requirements specific to their jurisdiction, it is expected that the SOLA Desktop will be the primary SOLA component requiring customisation to satisfy the needs of the agencies. This may extend to replacement of the SOLA Desktop with a custom client application. Where heavy customisation or replacement is required, the SOLA Desktop will act as a reference implementation providing working examples of the design and implementation patterns best suited for use with SOLA.

2.2 Services Layer

SOLA implements a web services based architecture that is consistent with the principles of Service Oriented Architecture (SOA)². It includes eight SOAP based web services implemented using the Java JAX-WS technology backed by eleven Enterprise Java Beans (EJBs) hosted using Glassfish and Metro.

The SOLA Desktop can use the SOLA Web Services independently or combined in order to support land administration business processes. All of the services are designed to be interoperable and available for reuse by other systems. Of note, Metro provides support for WS-* standards and is interoperable with .NET Windows Communication Foundation (WCF) technology.

The EJBs encapsulate the main business logic for SOLA and have been implemented using the EJB 3.1 specification. This specification provides several improvements³ over earlier EJB specifications and is intended to be light weight and functional.

2.3 Data Layer

The Data Layer persists SOLA data into a PostgreSQL database. The structure of the SOLA Database is based on the data storage requirements implied by the Land Administration Domain Model (LADM - draft ISO standard 19152) although extensions and adjustments have been included to support the function requirements of SOLA. The database contains multiple schemas with the data in each schema managed and maintained by a primary SOLA EJB. The PostGIS Database provides support for storage and manipulation of spatial data.

2.4 External Systems

External Systems identifies the systems SOLA integrates with. The SOLA Desktop can direct print jobs through to print servers on the land administration agent's network. Email notifications will be sent via the land administration agent's email server. Scanners can be configured to place imaged documents on the network or local file location so they may be picked up and linked to the relevant application.

¹ See section 4.1 in the SOLA Statement of Requirements

² http://en.wikipedia.org/wiki/Service-oriented_architecture#Principles

³ Convention over Configuration (CoC), Context and Dependency Injection (CDI), Portable Global JNDI Names, etc



3. Architectural Drivers

This section discusses the architectural drivers that have been identified for SOLA. The architectural drivers include the requirements that have architectural significance, constraints the system must comply with and architectural principles. The drivers provide rationale for key aspects of the SOLA architecture.

3.1 Requirements

3.1.1 Functional Requirements

The following table identifies and discusses the functional requirements of SOLA that have architectural significance.

Requirement	Description	Significance
Case Management	<p>The system will provide case management facilities for client initiated applications and requests. (FN – 1 to 3, 13 to 15, 17, 18, 21, 24, 25, 32 to 35, 38, 50 to 58, 64, 78, 80 to 82).</p> <p>Each land administration agency will have its own procedures and processes for managing land administration. The procedures and processes may involve multiple parties. (Generic Land Administration Process, FN – 49, 54, 78).</p>	<p>SOLA to implement a Case Management Service that will support the generic workflows for maintaining and managing case details and associated data.</p> <p>SOLA to extend the Land Administration Domain Model (LADM) as necessary to capture associations between the elements that comprise a case.</p> <p>SOLA Desktop to support the Generic Land Administration Process. Customisation of the SOLA Desktop by each land administration agency is to be expected.</p>
Search	<p>The system will provide facilities to search and display land administration records. (FN – 4 to 8, 19, 46, 47, 65 to 67, QL – 24, 36, 37).</p>	<p>Limited requirement for full text searching as data will be held as tabulated / structured text or in scanned image form.</p> <p>SOLA to implement a Search Service to perform structured text searches or combined structured text and geospatial searches using PostgreSQL / PostGIS database queries.</p>
Spatial Capability	<p>The system will provide a spatial capability sufficient to search, view, create and modify spatially defined cadastral objects. (FN – 4, 10 to 12, 36, 37, 39, 40, 42 to 45, 65 to 68, 71 to 74).</p> <p>The system will also support relevant technical standards for geographic information. (QL – 28).</p>	<p>SOLA to use a spatially enabled database (PostgreSQL / PostGIS), spatial library (Geotools) and a customized desktop GIS component to deliver rich spatial functionality.</p> <p>SOLA can also optionally accommodate a spatial map server (e.g. GeoServer).</p> <p>Support for Geographic Information standards (e.g. GML, WFS, LADM, etc).</p>



Requirement	Description	Significance
Rights Management	Each land administration agency will have its own data requirements for capturing and describing the legal rights that can be allocated to rights holders. (FN – 27 to 30, 56, 59 to 61).	<p>SOLA to implement an Administrative Service for maintaining and managing legal rights and associated data.</p> <p>SOLA to use the LADM to model core land administration concepts.</p> <p>SOLA to use a business rules engine to allow default rules to be modified to reflect agency requirements.</p> <p>Imaged documents are expected to cover the majority of data capture requirements however where additional Agency specific data requirements exist, these will need to be modelled explicitly in the SOLA database as part of the customisation process.</p>
Cadastre Management	The system will provide facilities to support maintaining the cadastre including the ability to capture versions of core cadastral records. (FN – 36, 37, 39 to 45, 58, 59);	<p>SOLA to implement a Cadastre Service for maintaining and managing cadastral processes and associated data.</p> <p>SOLA to use a spatially enabled database (PostgreSQL / PostGIS), spatial library (Geotools) and a customized desktop GIS component to deliver rich spatial functionality.</p> <p>SOLA to use the LADM to model core land administration concepts. LADM supports pseudo temporal data storage using time stamps and object versioning.</p>
Business Rule Management	Each land administration agency will have its own laws and regulations applicable to land administration. (FN – 1, 3, 18 to 20, 39, 40, 44, 48, QL – 3).	<p>SOLA to separate business rules from application code via a Rules interface.</p> <p>Rules interface to support the use of a Declarative Business Rules Engine (Drools Expert) or SQL statements for rule definition and execution.</p> <p>Agency specific requirements and/or changes to laws and regulations can be incorporated without requiring significant modifications to SOLA.</p>
Document Archive	The system will provide facilities to store and manage both generated and imaged documents. (FN – 17, 23 to 25, 32 to 35, 38, 45, 50, 53, 57, 64, 74).	SOLA to implement a basic Document Archive Service to support storage and retrieval of generated and imaged documents from the SOLA database.



Requirement	Description	Significance
Document Generation, Notification and Reporting	<p>The system will support generation of official extracts and documents suitable for reporting and external correspondence (e.g. Case Management notifications, etc). (FN – 1, 3, 9, 23, 50, 53, 57, 62, 69, 76, 77, 83, 84).</p> <p>The system will be capable of emailing generated correspondence (QL – 30)</p>	<p>No requirement for user to edit documents once they are generated and documents must be portable therefore documents to be generated in PDF format.</p> <p>SOLA Desktop to use the open source JasperReports component to generate PDF documents and reports.</p> <p>JavaMail API to be used for sending emails.</p>
Document Imaging	<p>The system will provide facilities to assist the scanning / imaging of documents that support land administration transactions. (FN – 17, 24, 25, 31 to 34, 38, 64, 95, QL – 27).</p>	<p>No direct integration between SOLA and scanning hardware is required.</p> <p>Document Archive Service to be extended to make use of a image manipulation library (Apache Commons Sanselan) and PDF library (PDF-Renderer) to generate thumbnail and preview images based on documents that have been scanned and placed in an accessible file system location.</p>
Printing	<p>The system will provide facilities to support printing of both generated and imaged documents. (FN – 1, 3, 9, 10, 23, 35, 50, 53, 57, 63, 69, 70, 85).</p>	<p>SOLA to provide printing facilities via the SOLA Desktop.</p>
Electronic Data Interchange	<p>The system will support transfer of data to or from the cadastral database. (FN – 39, 40, 71 to 73, QL – 29).</p>	<p>SOLA to provide spatial capability to support data extracts by Area of Interest.</p> <p>SOLA to extend the Cadastre Service to provide support for LandXML and CSV import and export.</p>
Auditing and Logging	<p>The system will maintain the integrity of all records and apply appropriate logging and quality control measures. (FN – 2, 6, 8, 10, 16, 70 to 74, 76, 77, 79, 83, 83, QL – 33).</p>	<p>SOLA to use Log4J to provide configurable logging options for system debug information and exceptions.</p> <p>SOLA to use application level auditing for nominated system functions/transactions using Context and Dependency Injection (CDI) Interceptors.</p> <p>SOLA to use trigger based audit trails at the Data Layer to track all data changes including those made directly to the database.</p>



Requirement	Description	Significance
System Administration	The system will provide features to support system administration, security and configuration. (FN – 10, 75 to 77, 83, 84, 88 to 94).	<p>SOLA to implement a System Service to support administration related updates and changes.</p> <p>SOLA to provide SOLA Administration section in SOLA Desktop for code list changes, business rule maintenance and related system configuration data.</p> <p>The SOLA Security Model is based around the capabilities of JEE Security and WS-* standards. Refer to the section 10 Security for details.</p>
Access to Property and Cadastre Information	The system will provide facilities to allow the public to request and/or obtain information on land administration records. (FN – 65 to 74)	SOLA to implement the SOLA Web Application to support deployment as a public counter service available from land administration offices. This web application could also be deployed on the internet to serve general public enquiries

3.1.2 System Qualities

The following table identifies and discusses the qualities of the SOLA (i.e. non functional requirements) that have architectural significance.

Quality	Description	Significance
Security	<p>The system will include security measures for user authentication, authorization, confidentiality and integrity. (QL – 31, 32).</p> <p>The system will ensure the integrity of all data stored. (QL – 1, 2).</p> <p>Periodic back-up of system data will be supported to avoid data loss. (QL – 9)</p>	<p>The SOLA Security Model is based around the capabilities of JEE Security and WS-* standards. Refer to the section 10 Security for details.</p> <p>SOLA to use the Java Transaction API (JTA) to support Distributed Transactions (a.k.a. 2 Phase Commit).</p> <p>SOLA database to use the UNICODE (UTF-8) character set to ensure accurate storage of multi-lingual data.</p> <p>SOLA to use an industry standard database with online backup capability (PostgreSQL).</p>
System Performance	User volumes and performance targets for the system (QL – 34 to 39)	<p>Separate Search Service to improve performance of searches that would otherwise require coordinating and managing the output from several service calls.</p> <p>Transaction logging to include start and end times for performance reporting.</p> <p>Refer to the section 9.3 Performance for details.</p>



Quality	Description	Significance
Maintainability	The system will comply with development techniques and standards that encourage high quality, maintainable software (QL – 10, 11, 13 to 16, 23)	Adhere to coding standards. Use code metrics tools (e.g. Sonar) to regularly gauge the system complexity. Use JUnit for automated unit tests. Use Fitnesse for automated system tests. Use Subversion of source control. Use JavaDoc for API documentation. Use Log4J to capture exception information for later analysis and JavaMail API to forward exception details via email.
Scalability	The system must be capable of being deployed in load balanced and /or failover configurations (QL – 8)	Use JEE 6 and Glassfish Application Server to support distributed deployment and load balancing of SOLA Services if required. SOLA Web Services and EJBs designed as stateless to support scaling out of services. SOLA to use an industry standard database technology (PostgreSQL) that supports scaling up, replication and fail over configurations.
Deployment	<p>The system must be capable of supporting a centralized as well as a hub-and-spoke deployment models (QL – 17)</p> <p>The system must be portable across major operating systems. (QL – 19 to 22, 24)</p> <p>The system must provide automated installation packages for major system components. (QL – 12)</p>	<p>Use Global Unique Identifiers (GUIDs) for record ids to avoid id clashes when using replication.</p> <p>Use Slony-I for enterprise level replication. Slony-I supports Master-Slave replication, but may not support Hub-and-spoke model indicated by QL – 19. Further investigation required.</p> <p>Use Java as the development language to support portability requirements.</p> <p>Enterprise Archive (EAR) used for deployment of SOLA Services and EJBs into the Services Host.</p> <p>Java Web Start used for deployment of the SOLA Desktop to minimise installation and configuration requirements on end users computers.</p>



Quality	Description	Significance
Data Migration	The database to support ETL tools to assist data migration activities. (QL – 40)	<p>SOLA to use an industry standard database (PostgreSQL) which is supported by a range of open source⁴ as well as commercial ETL tools.</p> <p>The Land Administration Data Model (LADM) to be used as the basis for the SOLA database and support compatibility with other LADM based products.</p>
Localization	The system will be capable of supporting multiple languages (QL – 1)	<p>Use of Java Locale to identify the localization parameters.</p> <p>Use of Java Resource Bundles to provide localized translations for SOLA Desktop screen labels and static text.</p> <p>Messaging Utility to centralize message processing and provide localized translations for SOLA generated messages via Java Resource Bundles.</p> <p>SOLA reference data display values capable of supporting multiple translations for display to the user.</p> <p>Support for calendar localization to be investigated.</p>
Low Power Supply Environments	Some land administration agencies operate in environments where power supply is not guaranteed and power outages are a common occurrence. (QL – 25)	<p>Hardware configuration to include an Uninterruptable Power Supply (UPS) to allow sufficient time for controlled shutdown of systems.</p> <p>Performance Testing of SOLA to be undertaken to ensure CPU and related resource usage is minimised.</p> <p>SOLA to incorporate an alert system that notifies users when significant actions occur such as transition to UPS power. This is a proposed feature that requires further investigation.</p>

⁴ Example open source ETL tools supporting PostgreSQL include; Pentaho Kettle, GeoKettle, Talend Open Studio and Talend Spatial Data Integrator.



3.2 Constraints

The following table identifies and discusses the main architectural constraints that SOLA must comply with.

Constraint	Description
Open Source	To achieve the goals of the SOLA project, the software must be free/libre and open source. Any software components or standards used for SOLA (e.g. database, GIS software, etc) must be non-proprietary and compatible with the Modified BSD license ⁵ . (QL – 18)
Portability	SOLA must be fully transferable to a wide range of major hardware or operating system platforms. The development language and supporting components used for SOLA must all comply with the portability requirements. (QL – 19 to 22, 24).

3.3 Principles

Architectural principles are used to guide architectural and high level design decisions to ensure the architecture is consistent. Each architectural principle is described via a standard pattern as follows

- Principle name
- Description
- Rationale / Benefits
- Implications

From consideration of the architectural requirements and constraints the following architectural principles have been established for SOLA.

- Leverage open source components and non proprietary industry standards
- Use modular loosely coupled components where possible
- Use and Enterprise Application Framework

Leverage open source components and non proprietary industry standards	
<i>Description:</i>	<p>Technology and infrastructure selection decisions will be based upon the availability of proven open source components and non proprietary industry standards.</p> <p>Custom solutions should only be used when open source components and/or non proprietary industry standards are unavailable, unproven or insufficient to satisfy project requirements.</p> <p>Proprietary components and standards should be avoided unless they are ubiquitous (e.g. Windows operating system).</p>

⁵ Open Source License selected for the SOLA Development Snapshot.

**Leverage open source components and non proprietary industry standards**

<i>Rationale / Benefits:</i>	<p>Compliance with standards avoids a dependence on the skills to build or customize and maintain a custom solution. Such skills tend to be in short supply.</p> <p>Facilitates and simplifies interoperability (technical and functional) between differing systems.</p> <p>Proprietary components and standards will limit the flexibility of SOLA and likely impose technical and financial constraints on the land administration agencies wishing to use it.</p> <p>Complies with the SOLA Open Source architecture constraint.</p>
<i>Implications:</i>	<p>Use of open source components (e.g. Postgresql, PostGIS, Glassfish, Metro, etc) as well as compliance with recognised non proprietary standards (e.g. WS-*, WFS, LADM, etc).</p>

Use modular loosely coupled and interoperable components where possible

<i>Description:</i>	<p>IT solutions should be engineered with a bias towards using highly discrete, loosely coupled and interoperable components.</p>
<i>Rationale / Benefits:</i>	<p>Loose coupling reduces the complexity of a system of interacting components. It allows making internal changes to one component without affecting other components. It improves availability and stability of the system since problems with one component are less likely to impact other components.</p> <p>Components can be reliably changed more quickly than otherwise would be the case. Functional scope of the components is reduced which in turn reduces their complexity.</p> <p>Interoperability enhances opportunities for reuse, reduces costs by reducing duplication of effort and reducing integration complexity.</p>
<i>Implications:</i>	<p>Use of layering (Presentation, Services and Data) to decompose the system into manageable blocks.</p> <p>Use of a web services based architecture and Metro to support WS-* compliant interfaces (QL – 26) minimizing the barriers for integration with other systems.</p> <p>Use of stateless EJBs to encapsulate business logic and Dependency Injection to allow alternative EJB implementations to be deployed and used in preference to the default EJB implementations.</p> <p>Use of rules engine to separate business rules from application code and to centralise and simplify rule definition and maintenance.</p> <p>Use Separated Presentation design pattern and beans binding in the presentation layer to separate the user interface from model data and controlling logic (i.e. beans) improving maintainability and testability.</p> <p>Use of Domain and Repository patterns to separate business logic from ORM implementation details.</p>



Use an Enterprise Application Framework (EAF)	
<i>Description:</i>	An EAF provides services and support for common enterprise system mechanisms such as persistence, security, messaging, UI presentation, etc.
<i>Rationale / Benefits:</i>	<p>Typically the EAF will comply with recognised standards ensuring interoperability and significantly reduce the overall coding effort required to produce reliable, scalable and performant enterprise applications.</p> <p>EAFs are usually well supported through Integrated Development Environment (IDE) tooling.</p> <p>Removes the need to evaluate and integrate a large range of components each responsible for achieving specific mechanisms (e.g. persistence framework, UI framework, etc).</p>
<i>Implications:</i>	<p>Use Java Enterprise Edition 6 as the EAF for SOLA. (QL – 20 to 22, 24)</p> <p>Use Glassfish 3 and Metro as the application server platform for SOLA.</p> <p>Note that JEE is a mature standard which is portable across a number of application server vendors and will have the largest vendor support base in future. The JEE standard documentation and learning path is also of a higher quality than alternative languages and other Java based Enterprise Application Frameworks. Coupled with broad vendor support this will further enable the land administration agencies that adopt SOLA to enhance and maintain their own SOLA implementations.</p>



4. Architectural Mechanisms

This section presents the architectural mechanisms of the SOLA and identifies how these will be incorporated during design and implementation. Note that architectural mechanisms represent common solutions to frequently encountered architectural problems and are often used to realise architectural requirements.

4.1 Analysis Mechanisms

The following table describes the analysis mechanisms applicable to SOLA.

Analysis Mechanism	Description
Persistence	The means to make an element persistent (i.e. save an object's state across multiple executions of the application).
Communication	The means by which distributed processes communicate.
Business Logic Encapsulation	The means by which business logic is encapsulated in the application to enable consistent reuse of the logic while also remaining loosely coupled and maintainable.
Transaction Management	The means by which ACID transactions are handled.
Security	Protect access to certain resources or information.
Auditing	Provide audit trails of system execution and access to system managed resources and information.
Business Rules	The means to manage and execute business rules.
Workflow	The means to manage and control the progress of key items as they are processed through the systems.
Error Management	Allows errors to be caught, propagated, managed and reported.
Document Generation & Reporting	The means by which the system will generate documents and reports.
User Interface	Provide facilities for windows presentation and user interaction.
Electronic Data Interchange	The means by which the system will exchange data.
Notifications	The means by which the system will generate user notifications.
Printing	The means by which the system will print system generate documents.
Geographical Information Systems (GIS)	The means by which the system will support storage, viewing and manipulation of GIS dat.
Image Manipulation	The means by which the system will support manipulation of images.
Localization	The means by which the system will support localization of the product to a specific locale and or provide runtime support for multiple locales.



4.2 Analysis Mechanism Mappings

The following shows how the analysis mechanisms above have been mapped to design and Implementation mechanisms.

Analysis Mechanism	Design Mechanism	Implementation Mechanism
Persistence	Relational Database	PostgreSQL
	Object Relational Mapping (ORM)	Java Persistence Architecture API (JPA) and Hibernate Java Persistence Framework
	Data Replication	Slony-I
Communication	Inter Process Communication (IPC)	Java API for XML Web Services (JAX-WS), Glassfish Metro
Business Logic Encapsulation	Business Interfaces	Enterprise Java Beans (EJB), Dependency Injection
Transaction Management	Distributed Transactions	Java Transaction API (JTA)
Security	Authentication	Username Authentication with Symmetric Key and Trusted Subsystem Model, WS-*, Glassfish, Metro
	Authorisation	Role Based Access Control (RBAC), Glassfish Roles and Groups, Declarative Authorisation
	Communication	WS-* protocols, Symmetric Data Encryption
Auditing		DB Triggers, Log4J
Business Rules	Business Rule Service	Declarative Business Rules Engine (BRE) – Drools Expert, SQL Queries
Workflow	State Transition Modelling	Custom state/status attribute on key entities
Error Management	Structured Exception Handling and Logging	Log4J logs, Web Services Fault Contract
Document Generation & Reporting	Report Generator/Tool	JasperReports
User Interface	Desktop Client	Java Swing, MVC pattern, beans binding
Electronic Data Interchange	Flat file and XML	Comma Separated (CSV), LandXML
Notifications	Email	JavaMail API, Simple Mail Transfer



Analysis Mechanism	Design Mechanism	Implementation Mechanism
		Protocol (SMTP)
	Document Format	Portable Document Format (PDF)
Printing	Client Side Printing	JasperReports Viewer
Geographical Information System (GIS)	Spatial Data Persistence	PostGIS extension for PostgreSQL
	GIS Library	Geotools
	Desktop GIS	Custom Swing component
Image Manipulation	Image Manipulation Library	Sanselan, PDF-Renderer
Localization	Screen Labels	Java Locale, Swing and Java Resource Bundles
	User Messages	Java Locale and centralized message processing utility using Java Resource Bundles
	Reference Data	Database function to obtain translations for display values
	Currency	Java Locale, currency / money utility
	Calendar	TBC



5. Use Case View

This section identifies those use cases or scenarios from the use case model that represent significant, central functionality of the system, address architectural risks or they have a large architectural coverage – that is they exercise many architectural elements, or if they stress or illustrate a specific, important point of the architecture.

5.1 Architecturally Significant Use Cases

The following use case diagram shows those use case(s) that are considered architecturally significant.

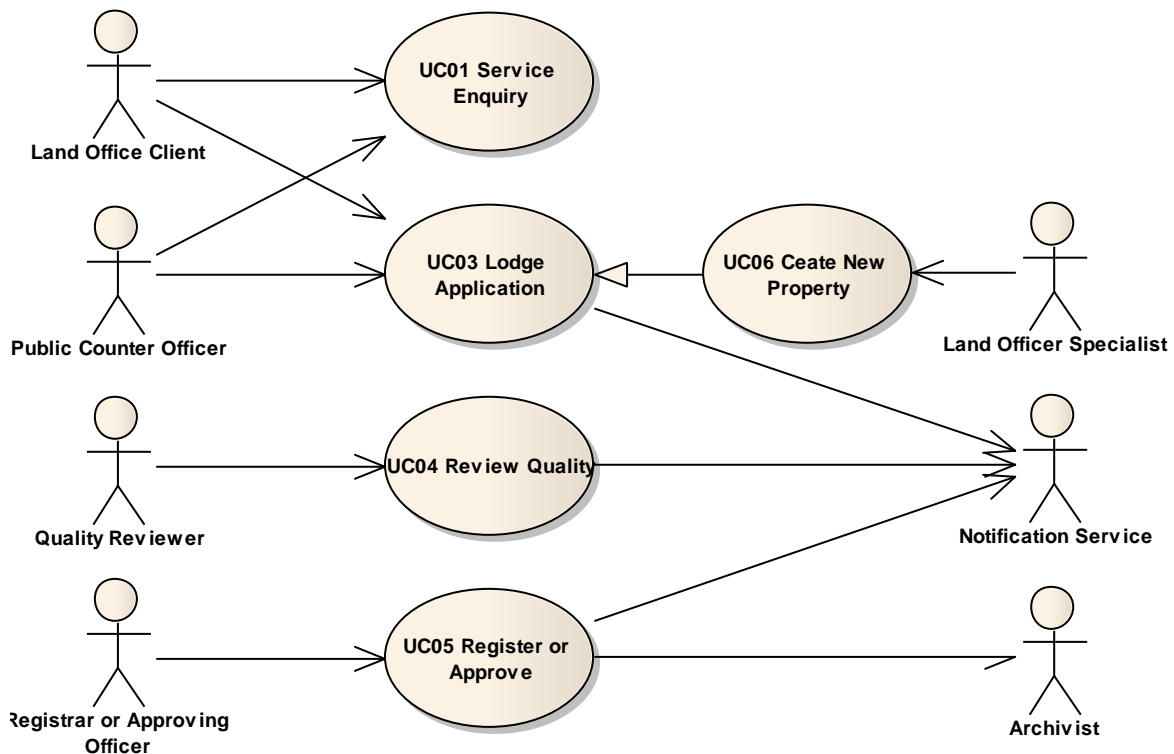


Figure 2 – Architecturally Significant Use Cases

The uses cases that are part of the SOLA but are not identified above are considered simpler and more straightforward examples of the architecturally significant use cases and are not represented explicitly in the Software Architecture Document. However the architectural approach and mechanisms will be the same for the simpler use case as for the complex use cases explicitly described here.

The following sections briefly describe the architecturally significant features for each of the use cases identified and include references to the architectural risks⁶ addressed by each feature.

5.1.1 UC01 Service Enquiry

An enquiry made by a land office client concerning the status of an existing service application or a general enquiry about the services provided by the land office and the requirements and fees pertaining to the service.

⁶ Refer to Appendix 3 – Architectural Risks for details.



Architecturally significant aspects of this use case include

- Search Records (FN 4) – Uses the Search Service to improve cross service search performance (Risk 4 & 7).
- View Cadastral Map (FN 10) – Uses the SOLA Map Viewer and Spatial Service to display the cadastral map (Risk 3, 4, & 6)

5.1.2 UC03 Lodge Application

The receipt and the recording of a land office service request details and any changes to rights or property definitions arising from the application. Initially this is completed by a land officer serving at a public counter in the land office and subsequently by a land officer working in the “back office”.

Architecturally significant aspects of this use case include

- Lodge New Application (FN 21) – Coordinates multiple EJB’s to save application details (Risks 4 & 7).
- Lodge Notice (FN 23) – Uses Jasper Reports to generate the Lodgement Notice (Risk 3)
- Application Main Documents (FN 24) – Uses the Digital Archive Service to support linking of imaged documents to the application (Risk 2).
- Link New Boundary Nodes to Existing Cadastre Nodes (FN 43) – Illustrates using the SOLA Map Viewer for spatial editing tasks (Risks 3 & 6)

5.1.3 UC04 Quality Review

The review of the recording of the application and any changes to rights or property definitions to ensure their correctness, that all required supporting documents have been provided and that these are consistent with the application. The “back office” land officer will conclude this review with a recommendation as to whether the application can proceed.

Architecturally significant aspects of this use case include

- Complete Quality Checklist (FN 48) – Uses the business rules engine to automatically verify the quality of the data supporting the application (Risk 3)

5.1.4 UC05 Register or Approve

The duly delegated land officer (typically registrar or approving surveyor) will further review an application and, if satisfied, will approve (a change to the cadastre) or register (an application for registration). At that time the proposed changes to rights or property definitions take the status of “current” and any former rights or property definitions take the status of “historic”.

Architecturally significant aspects of this use case include

- Register Transaction (FN 55) – Uses the Administrative Service and the Cadastre Service to apply status changes to land administration records that must be managed as one transaction (Risks 5 & 7).

5.1.5 UC06 Create New Property

Where an application is received to create a new property, details concerning the new property must be recorded, any existing rights either cancelled or transferred to the new property and the existing property records to be superseded identified. A new certificate is then issued for the new property (a certificate of title in jurisdictions where title registration is practised).



Architecturally significant aspects of this use case include

- Verify Draft Title Details (FN 59) – Uses the Cadastre Services to effect changes to spatially defined land administration records (Risk 3)

5.2 Use-Case Realisations

This section illustrates how the software will work by giving the use case (or scenario) realisations of the architecturally significant use cases and features. These use case realisations consist of sequence diagrams which model how the various design model elements of the system will interact to achieve the required behaviour.



5.2.1 UC01 Service Enquiry – Search Records (FN 4)

The Public Counter Officer enters search criteria into the Application Search Form, which may include application details as well as agent or contact person (i.e. party) information. The Search EJB executes a search using its entity manager⁷ and a JPA Named Query⁸ across both the Application and Party schemas. The results are returned to the Application Summary Bean which notifies the Application Search Form to refresh via a Property Changed event.

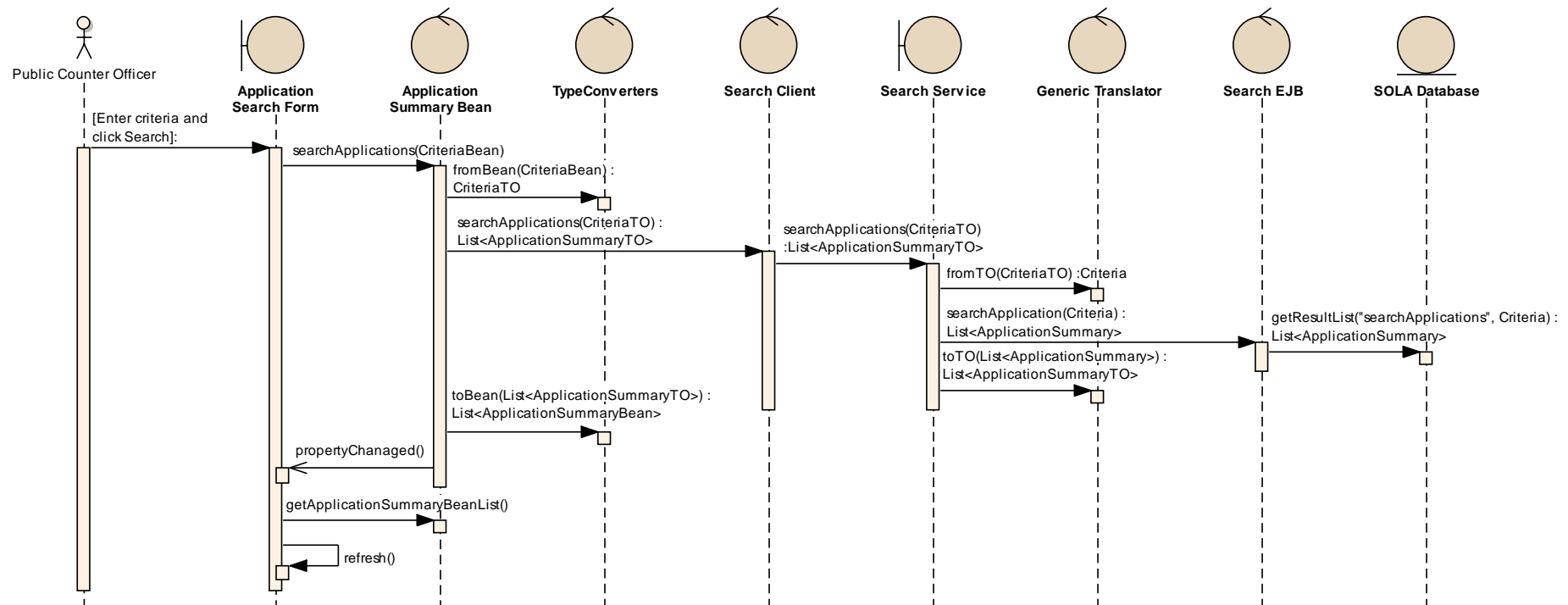


Figure 3 – UC01 Service Enquiry – Search Records (FN 4) Sequence Diagram A

⁷ <http://www.objectdb.com/api/java/jpa/EntityManager>

⁸ <http://www.objectdb.com/java/jpa/query/named>



The Public Counter Officer can view the details of a selected application in the Application Form. The agent, contact person, address and source details associated with the selected application are lazy loaded via the appropriate EJB and translated to equivalent TOs during the translation of the application in the Services Layer.

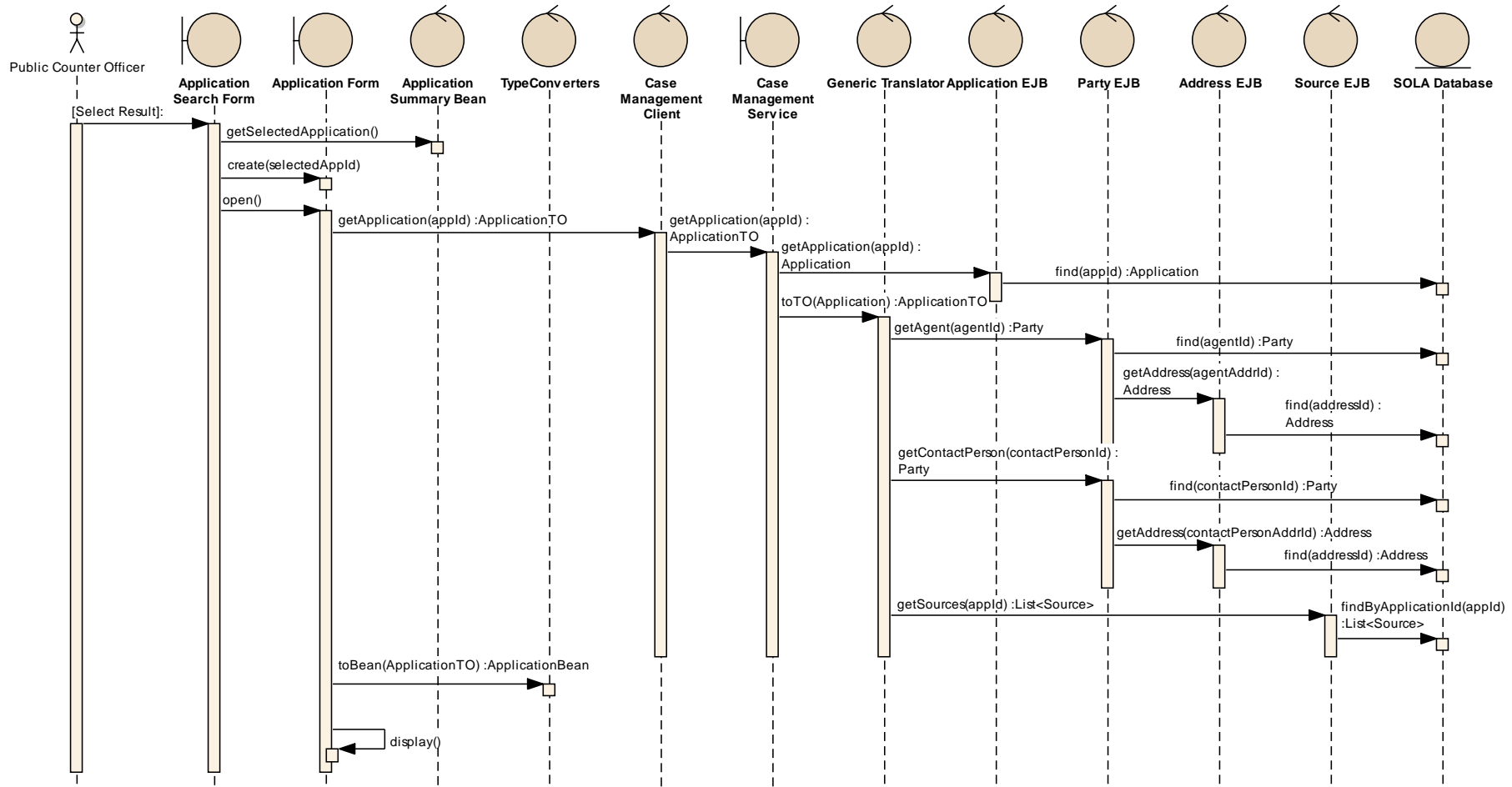


Figure 4 - UC01 Service Enquiry – Search Records (FN 4) Sequence Diagram B



5.2.2 UC01 Service Enquiry – View Cadastral Map (FN 10)

The Public Counter Officer selects the Map button to display the SOLA Map Viewer. The Starter is a Singleton that holds a reference to the Spatial Client through the Pojo Data Access class. Once the Map is configured, the data for each layer is loaded from the Spatial Service. The Result For Navigation entity is exposed by the Spatial Service to avoid the need to translate the navigation data into a TO.

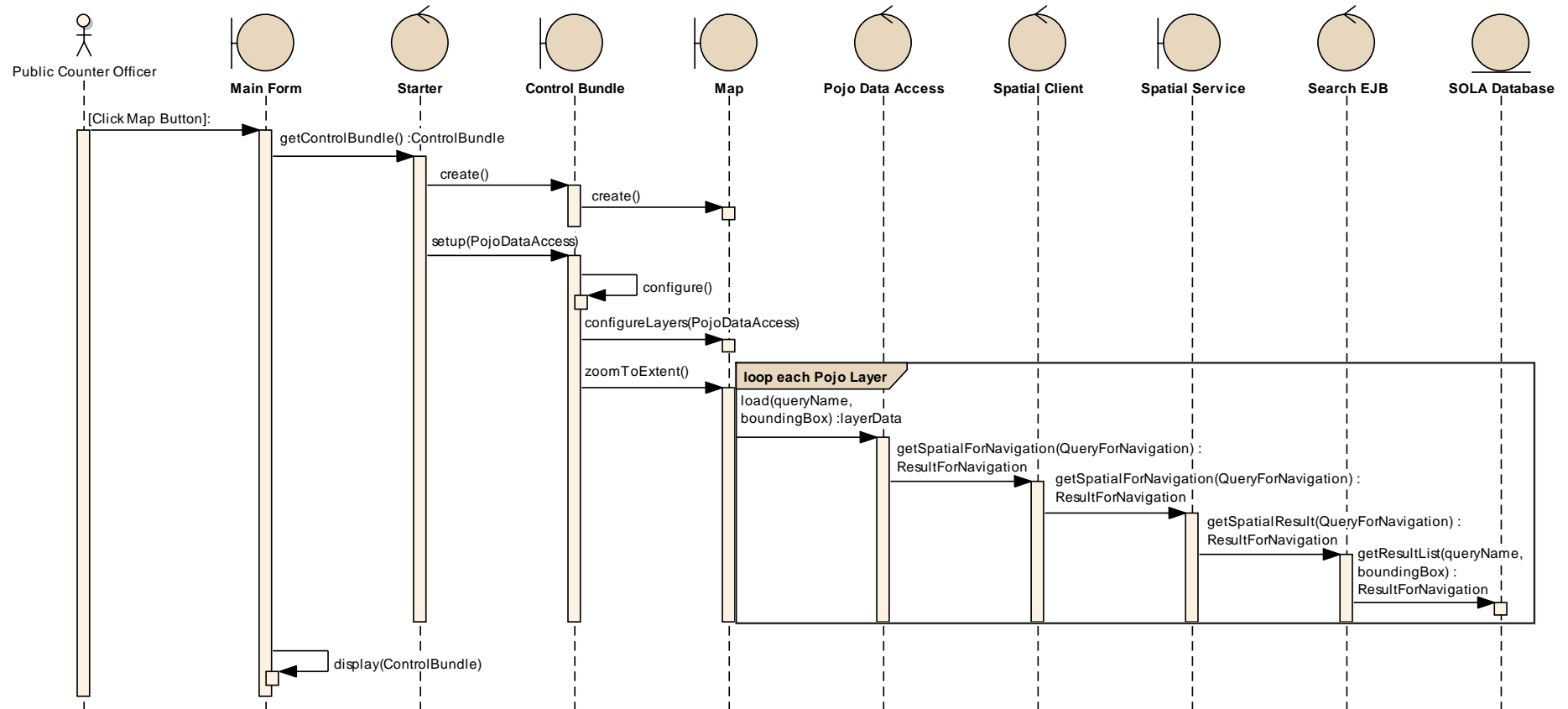


Figure 5 - UC01 Service Enquiry – View Cadastral Map (FN 10) Sequence Diagram



5.2.3 UC03 Lodge Application – Lodge New Application (FN 21)

Create Application uses the appropriate EJB to save new party, address and source details.

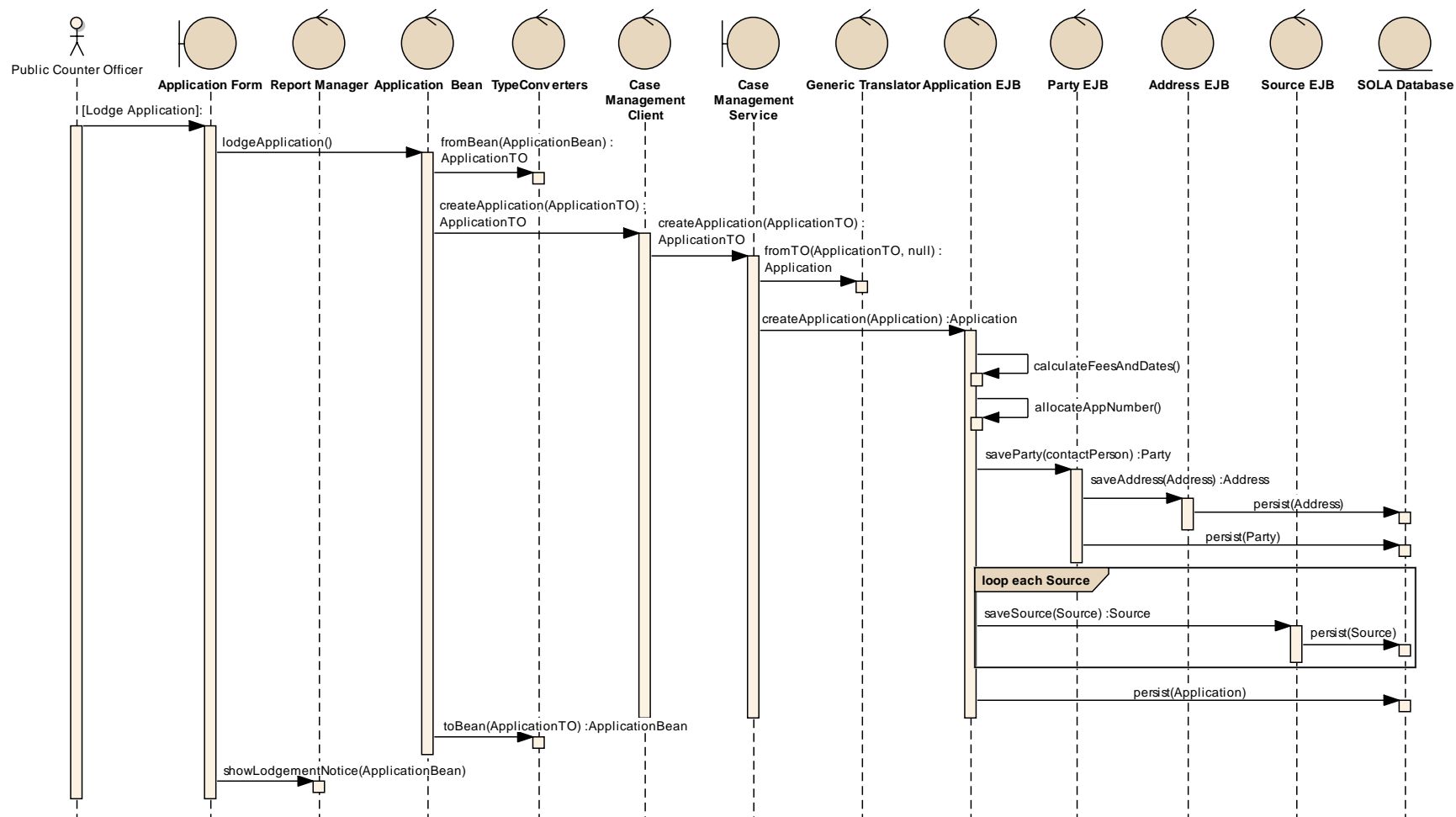


Figure 6 - UC03 Lodge Application – Lodge New Application (FN 21) Sequence Diagram A



Saving an application is similar to creating the application with the main difference that the application is first retrieved from the database the data from the Application TO is translated into the attached application. This ensures any entity properties that are not exposed through the Transfer Objects remain unchanged during JPA Merge / Persist operations (also see section 6.9.1 Abstract TOs).

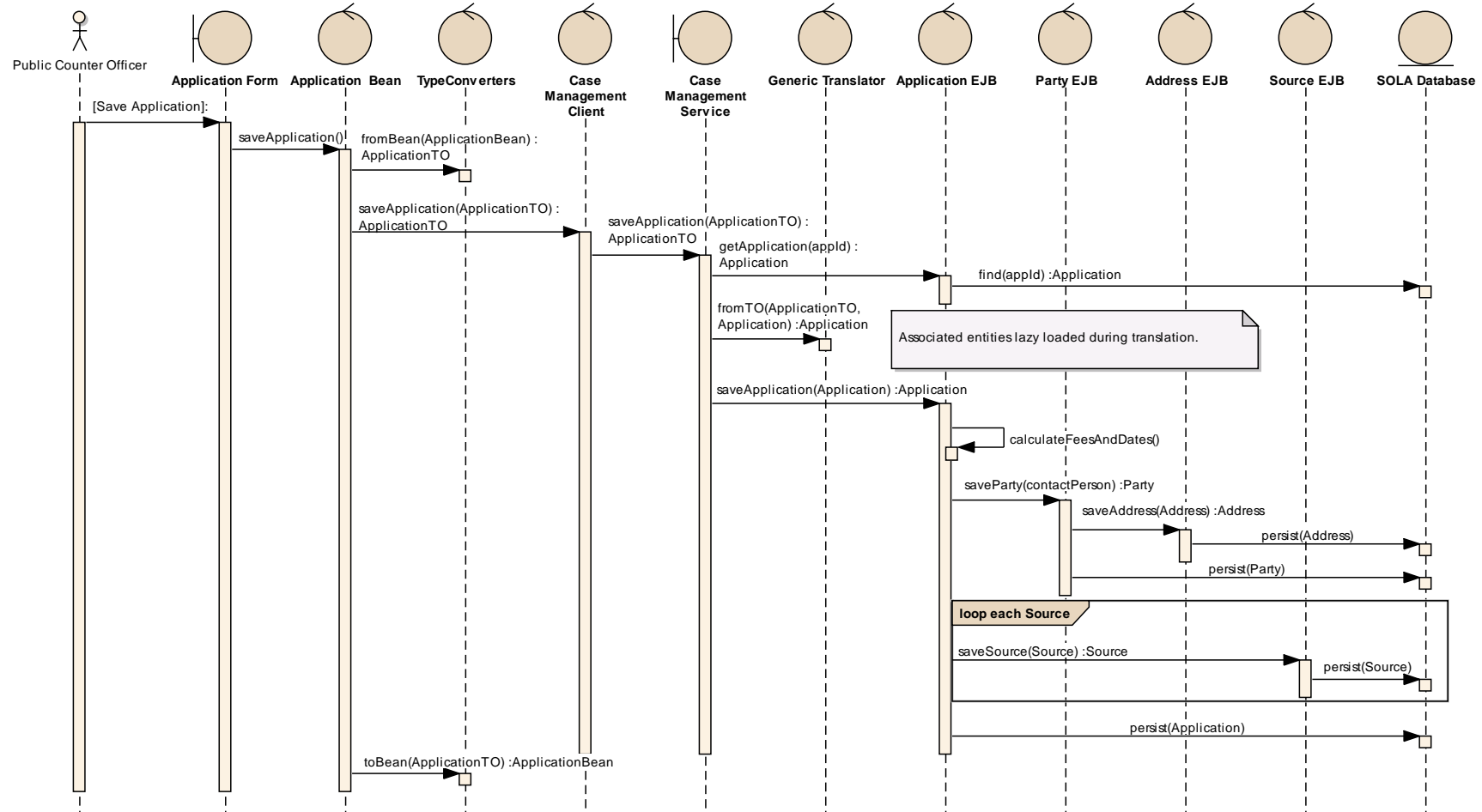


Figure 7 - UC03 Lodge Application – Lodge New Application (FN 21) Sequence Diagram B



5.2.4 UC03 Lodge Application – Lodge Notice (FN 23)

As part of lodging a new application, the lodgement notice is generated using Jasper Reports. The Lodgement Notice Template is filled with data from the Application Bean and displayed to the Public Counter Officer in the Jasper Viewer. The Public Counter Officer can optionally print the lodgement notice from Jasper Viewer.

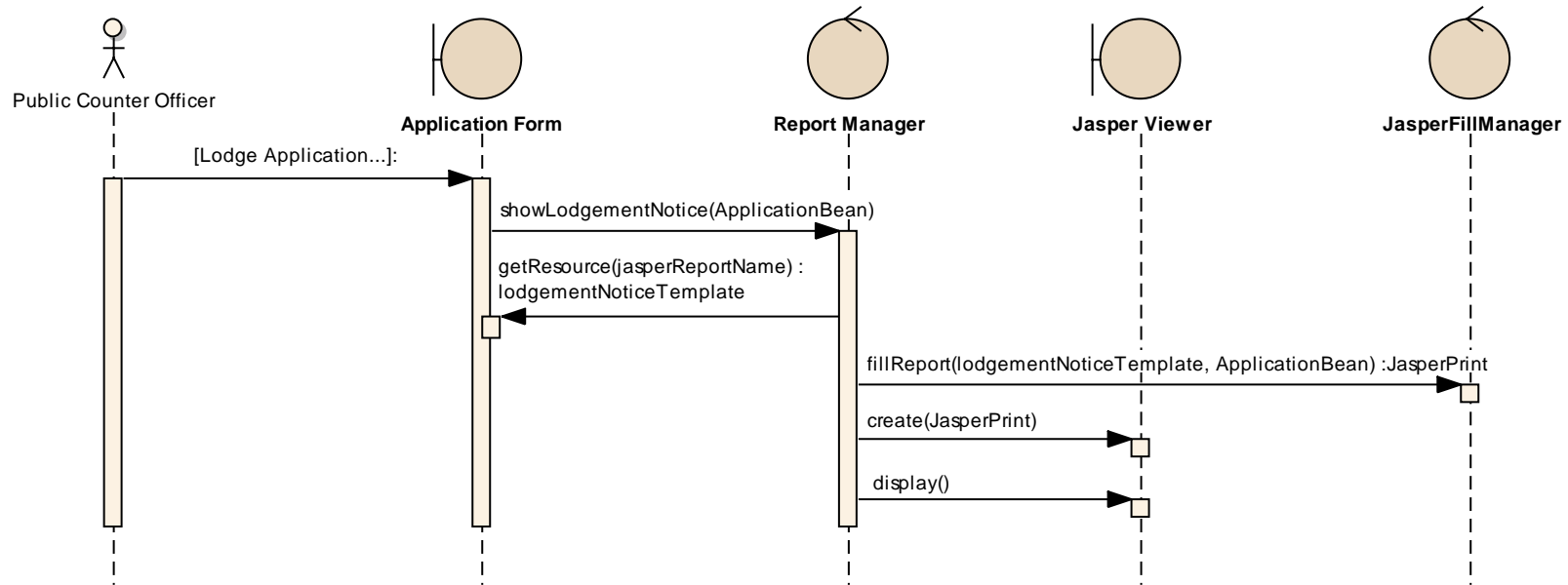


Figure 8 - UC03 Lodge Application – Lodge Notice (FN 23) Sequence Diagram



5.2.5 UC03 Lodge Application – Application Main Documents (FN 24)

When attaching a document, SOLA retrieves the list of files from the default file system location (an accessible network location) and displays basic file information (e.g. file name, etc) to the Public Counter Officer. The Public Counter Officer can choose to view a thumbnail of one of those files, or they can browse their local file system to locate the document to attach. This diagram illustrates retrieving the file information from the default file system location.

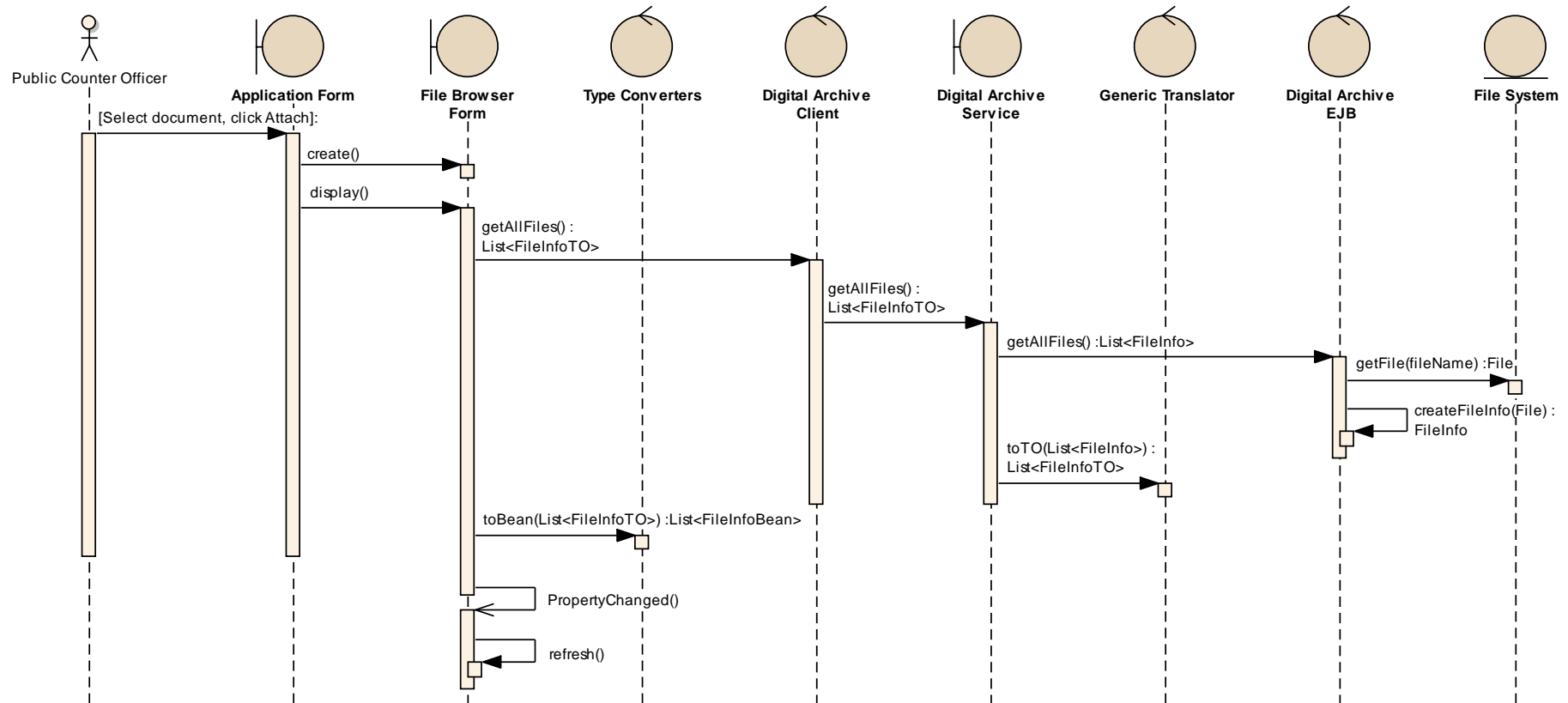


Figure 9 – UC03 Lodge Application – Application Main Documents (FN 24) Sequence Diagram A



If the Public Counter Officer selects one of the files, a thumbnail is generated so the officer can view the first page of the document and confirm it is the correct document to attach. The officer can also choose to open the document to view the full contents of the document (not illustrated in this diagram).

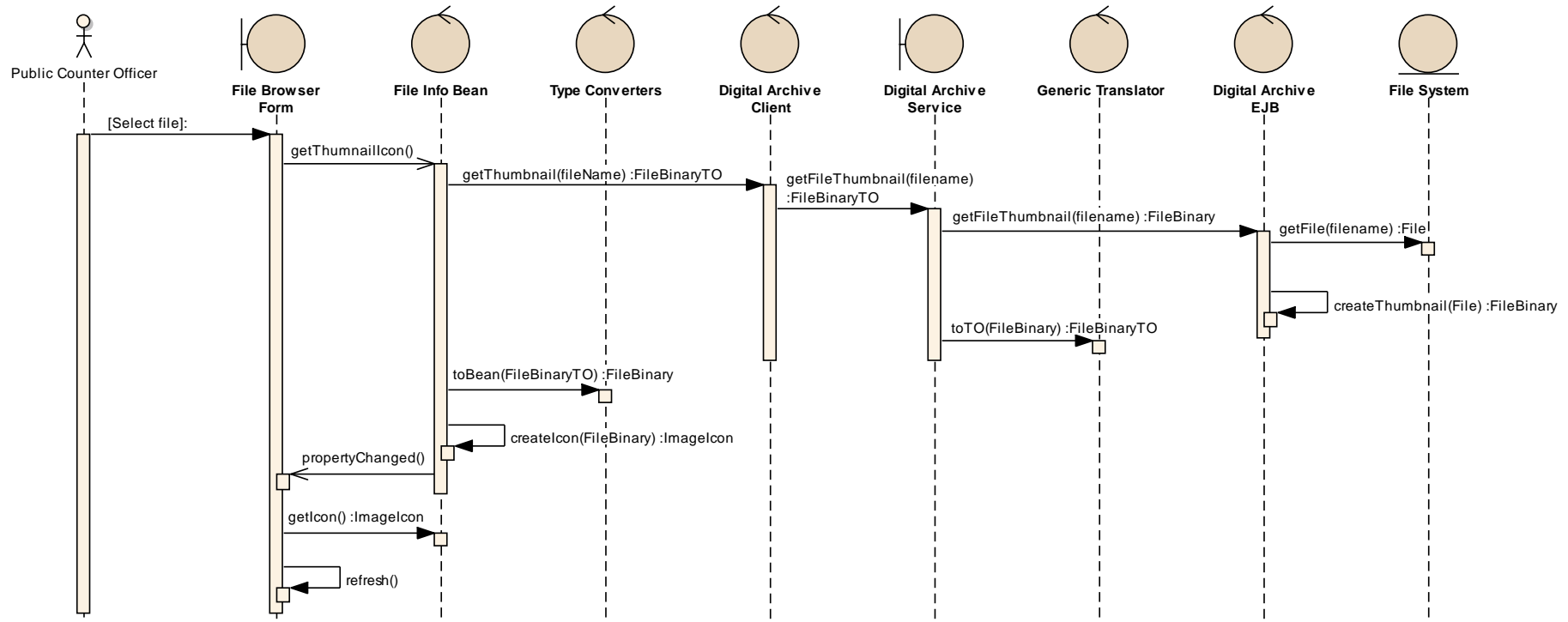


Figure 10 - UC03 Lodge Application – Application Main Documents (FN 24) Sequence Diagram B



When the Public Counter Officer confirms the document to attach, the document is saved to the SOLA Database.

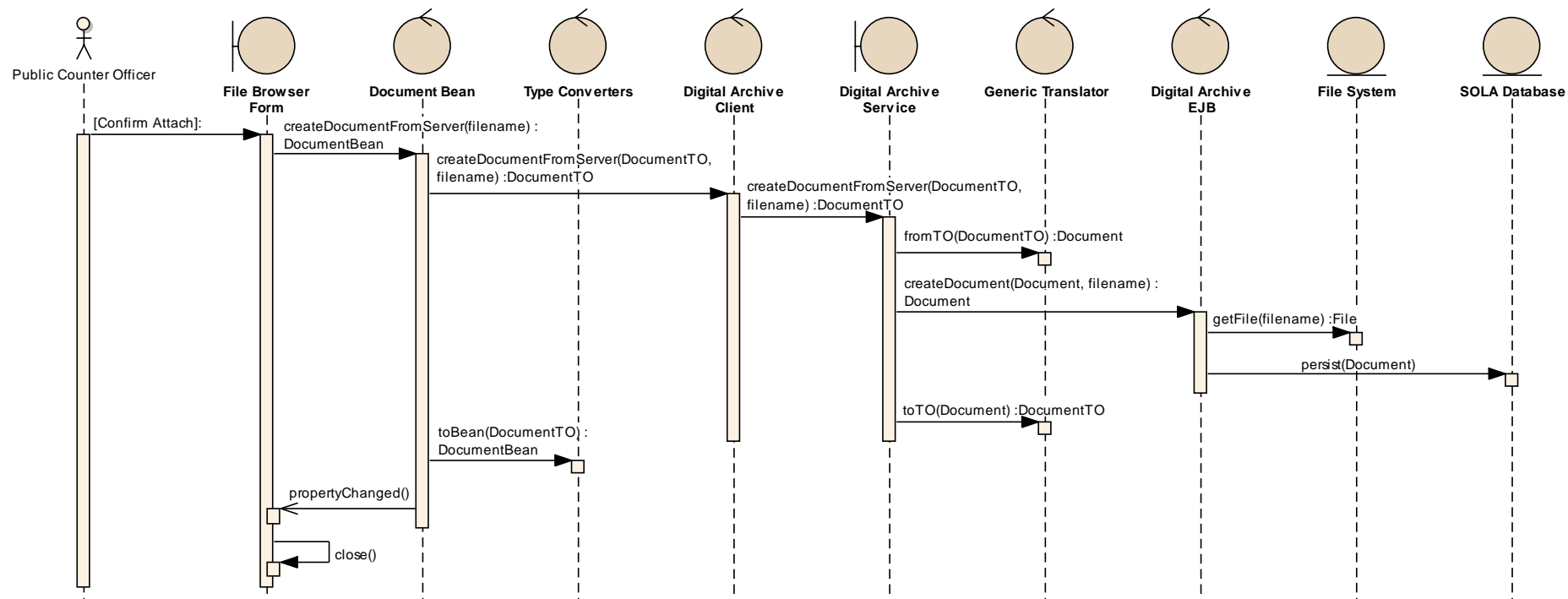


Figure 11 - UC03 Lodge Application – Application Main Documents (FN 24) Sequence Diagram C



5.2.6 UC03 Lodge Application – Link New Boundary Nodes to Existing Cadastre Nodes (FN 43)

To be completed.

5.2.7 UC04 Quality Review – Complete Quality Checklist (FN 48)

To be completed.

5.2.8 UC05 Register or Approve – Register Transaction (FN 55)

To be completed.

5.2.9 UC06 Create New Property – Verify Draft Title Details (FN 59)

To be completed.



6. Logical View

This section describes the architecturally significant parts of the design and its decomposition into packages and subsystems. For some significant packages the logical view is further decomposed into interfaces and classes.

6.1 Overview

The following package diagram shows the high level packages and elements that make up SOLA.

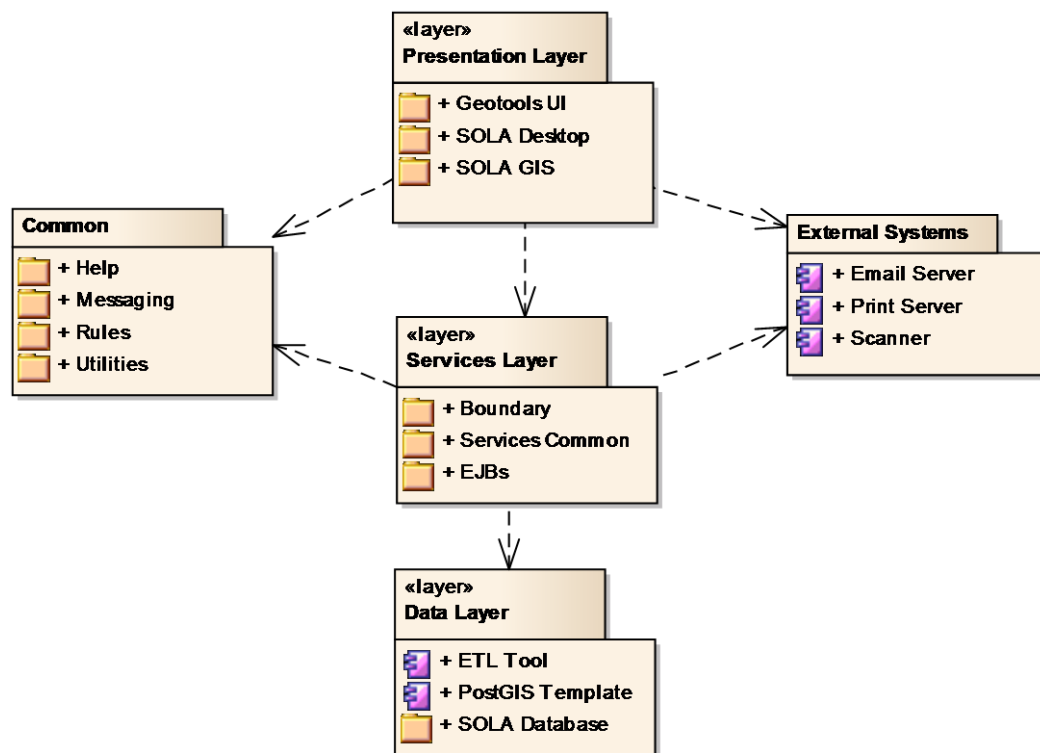


Figure 12 – SOLA Package Overview

The architecture of the SOLA has been designed using the layered architectural pattern. A two dimensional layering approach has been used to structure the software firstly by responsibility and secondly for reuse.

In the following sections each of the layers and high level packages are briefly described, along with any significant classes or components, their key responsibilities and the namespaces used for the packages. Note that all SOLA package names will stem from the base package name of `org.sola`.



6.2 Presentation Layer

The Presentation Layer encapsulates the components that end users will interact with, including the SOLA Desktop and the SOLA Map Viewer (SOLA GIS).

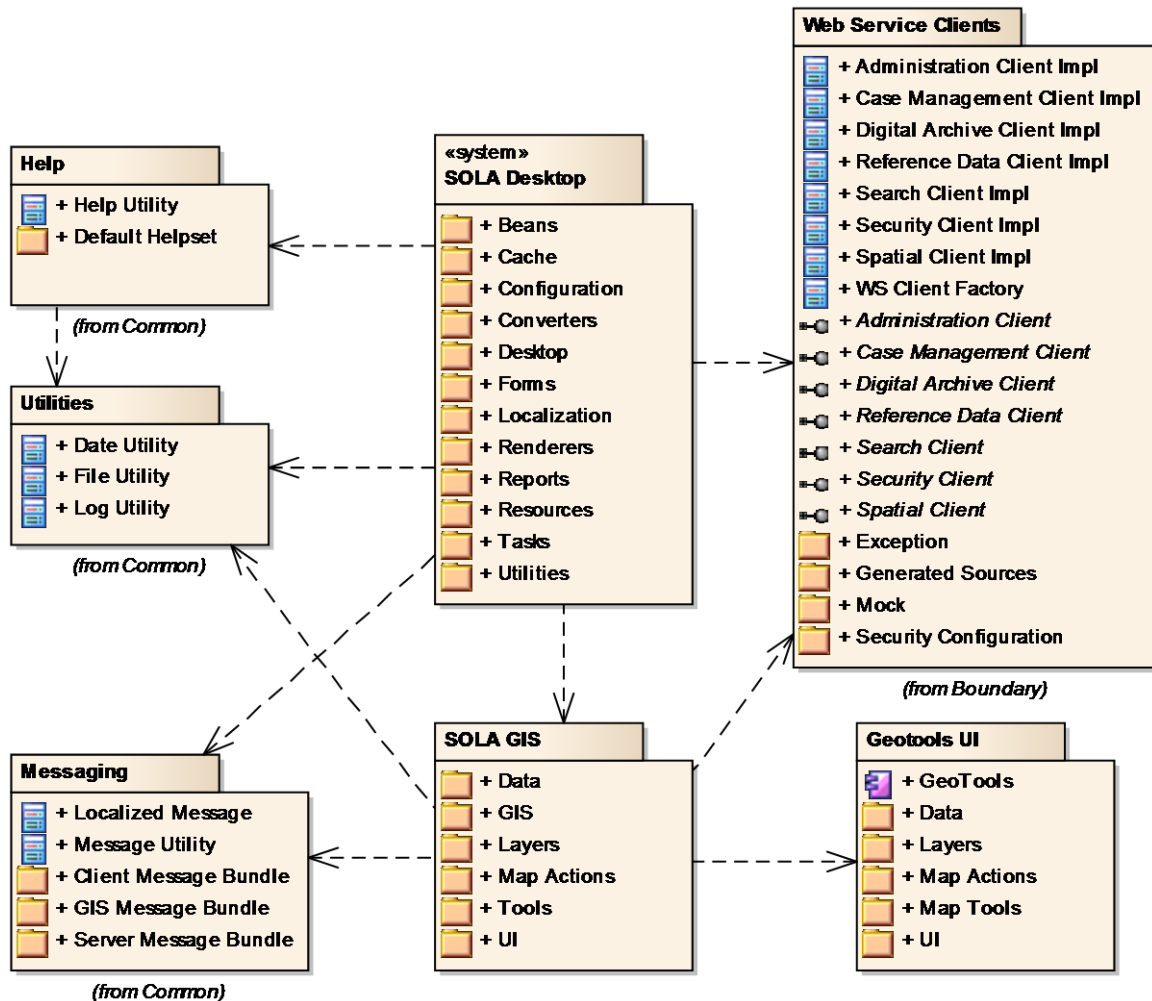


Figure 13 - Presentation Layer

Package names in the Presentation Layer stem from `org.sola.clients`.

6.3 SOLA Desktop

The SOLA Desktop is a Java Swing desktop application delivered to end users via Java Web Start technology to ensure minimal deployment and configuration overhead. The Development Snapshot version of the SOLA Desktop supports basic front office case management tasks including application lodgement, digital document preview and attachment, application search, application assignment and a custom spatial map viewer (SOLA GIS). The SOLA Desktop provides translations for both the English and Italian languages and can be configured to support additional languages.

The SOLA Desktop will continue to be extended to support the Generic Land Administration Process⁹. Given each land administration agency will have different workflow requirements specific to their jurisdiction, it is expected that the SOLA Desktop will be the primary SOLA component requiring customisation to satisfy the needs of the agencies. This may extend to

⁹ See section 4.1 in the SOLA Statement of Requirements



replacement of the SOLA Desktop with a custom client application. Where heavy customisation or replacement is required, the SOLA Desktop will act as a reference implementation providing working examples of the design and implementation patterns best suited for use with SOLA.

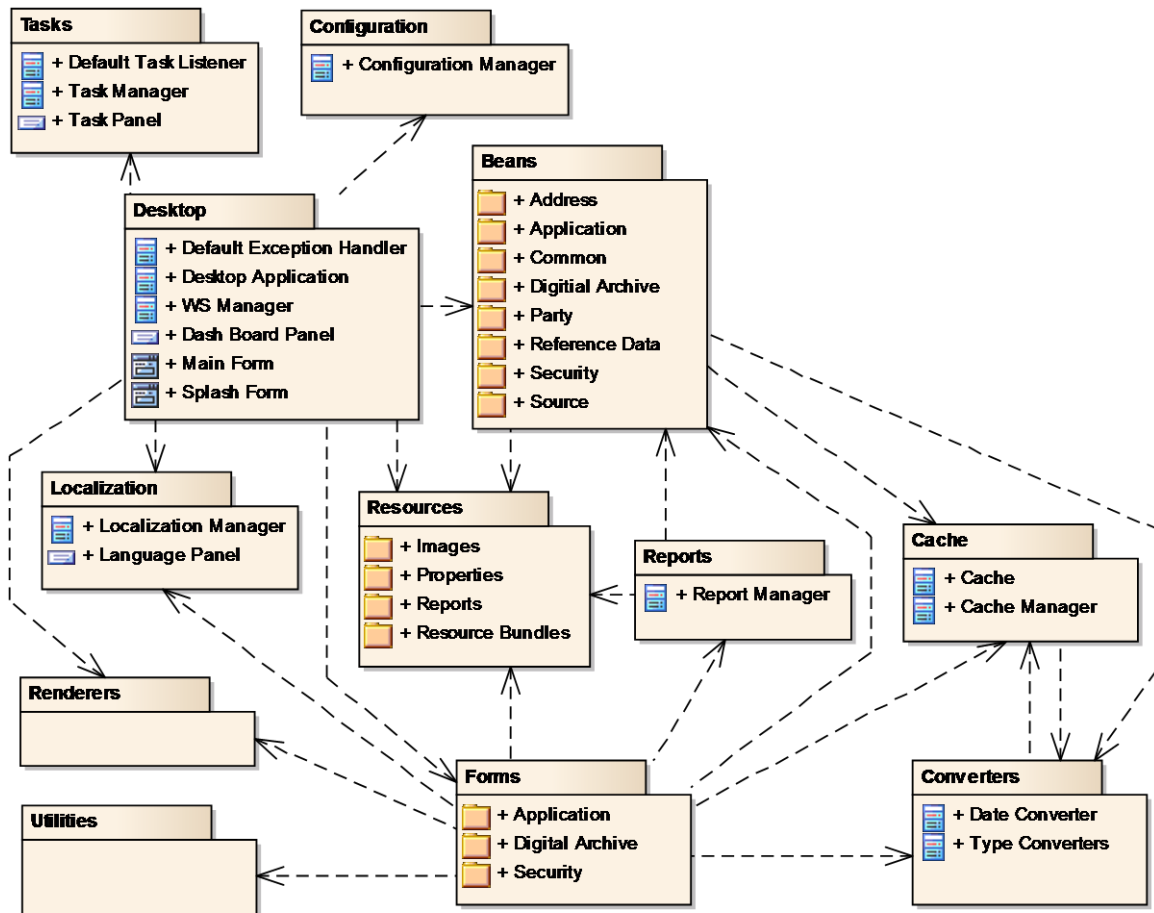


Figure 14 - SOLA Desktop

Package names in the SOLA Desktop stem from `org.sola.clients.desktop`.

6.3.1 Desktop

The Desktop package includes the main executable class for the SOLA Desktop (Desktop Application), the Main Form, Web Service Manager (WS Manager) and related application classes.

6.3.2 Beans

The SOLA Desktop implements the Separated Presentation¹⁰ UI pattern and the classes of the Beans package form the domain layer of this pattern. Better Beans Binding¹¹, which is a fork of the Beans Binding JSR 295 reference implementation, is used to notify the presentation of changes in the domain layer.

Note that JSR 295 was withdrawn in May 2011¹², however the latest versions of the NetBeans IDE continue to provide integrated support for beans binding¹³.

¹⁰ <http://martinfowler.com/eaDev/SeparatedPresentation.html>

¹¹ <http://kenai.com/projects/betterbeansbinding/pages/Home>

¹² <http://jcp.org/en/jsr/detail?id=295>



6.3.3 Forms

The Forms package contains the GUI Form classes of the SOLA Desktop. These classes are responsible for presentation of SOLA data only, consistent with the Separated Presentation UI pattern.

6.3.4 Reports

The Report Manager provides methods to generate and display SOLA reports using the Jasper Report Viewer. The Jasper templates for the SOLA reports are located in the reports folder of the Resources package.

6.3.5 Converters

This package provides utilities to translate data between the Transfer Objects exposed by the SOLA Web Services with the beans in the Beans package.

6.3.6 Localization

This package includes a standardised locale selection control and stores any locale selection made by the user in the Java Preferences store. When loading a form for display, the locale setting is used to determine the correct Java Resource Bundle¹⁴ to use as the source for the labels defined in the form.

6.3.7 Configuration

The Configuration Manager determines the URLs to use to connect the SOLA Web Services. The URLs are sourced from system properties in the case of SOLA Desktop Web Start or from a configuration file obtained from the SOLA Desktop JAR (for development).

6.3.8 Resources

The Resources package contains non-code resources used by the SOLA Desktop such as images, property files, resource bundles (for localization), and report templates.

6.3.9 Tasks

This package contains classes for running and monitoring the progress of background tasks. Background tasks can be used to execute long running actions and ensure the UI remains responsive to further user input. Currently background tasks are used to load the Unassigned and Assigned lists on the Dashboard. The user can choose to navigate to another screen using the menu options before these loading tasks are complete.

6.3.10 Cache

Cache provides simple caching facilities for system reference data and code lists.

6.3.11 Renderers and Utilities

These packages contain classes to assist the rendering of data values in other controls such as table cells and combo boxes. They also include general utility classes available for use across SOLA Desktop packages.

¹³ <http://netbeans.org/kb/docs/java/gui-binding.html>

¹⁴ <http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/>



6.4 SOLA GIS and GeoTools UI

The SOLA GIS and GeoTools UI packages implement the spatial components used for the SOLA Desktop. Following review of existing open source Java based GIS tools and components, GeoTools¹⁵ was selected as the basis for the SOLA GIS components. GeoTools is an open source GIS toolkit that consists of a number of libraries covering a broad range of GIS functionality. It provides implementations of Open Geospatial Consortium¹⁶ (OGC) specifications as they are developed and is used as the basis for other geospatial projects including GeoServer¹⁷ and uDig¹⁸. The original GeoTools project was started in 1996 and it remains an active project with the most recent release being July 2011 (version 8.0 M1).

The main GeoTools libraries used by SOLA are

- gt-swing – This library contains the basic Swing GUI components including a basic map control, status bar as well as action and tool classes to interact with the map control and map layers.
- gt-epsg-hsql – This library contains routines and classes to define coordinate systems.
- gt-wms – This library contains calls and routines to load WMS layers.

The SOLA GeoTools UI package has been implemented to extend the basic GUI components offered by the gt-swing library in a generic manner. The SOLA GIS package builds on the components in the GeoTools UI package to satisfy SOLA specific implementation requirements and has no direct dependency on GeoTools.

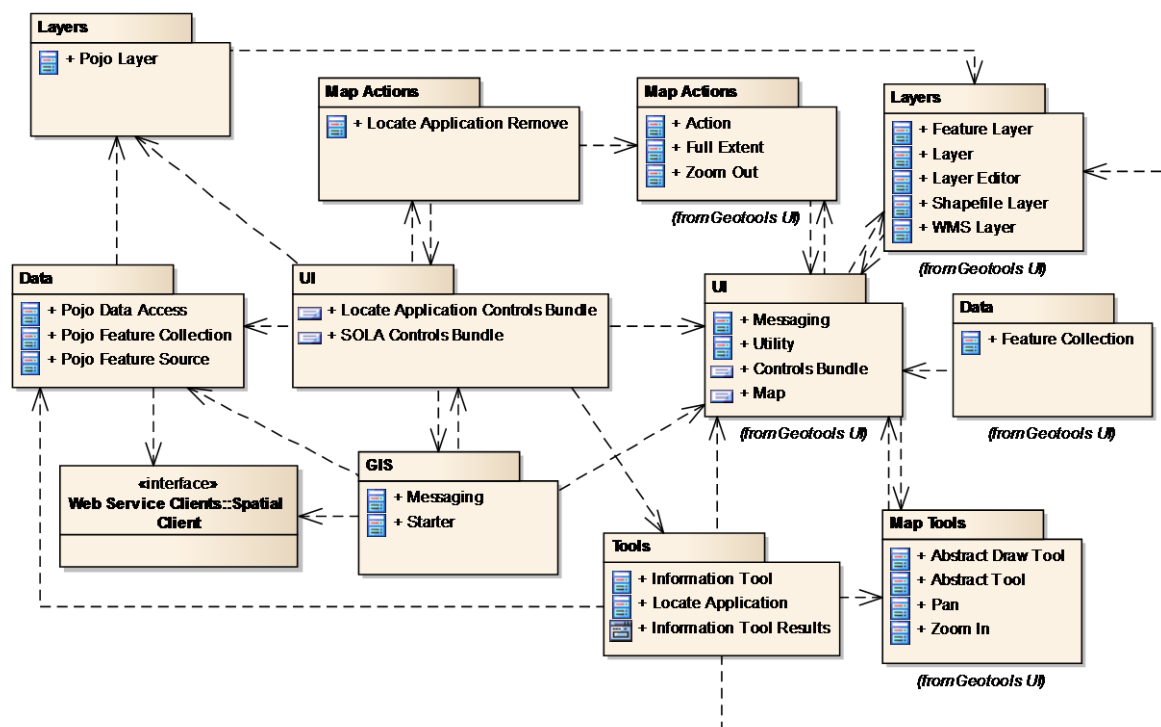


Figure 15 - SOLA GIS and GeoTools UI

¹⁵ <http://geotools.org/>

¹⁶ <http://www.opengeospatial.org/>

¹⁷ <http://geoserver.org/display/GEOS/What+is+GeoServer>

¹⁸ <http://udig.refractory.net/developers/>



The initial architecture for SOLA included GeoServer for serving geospatial data to the SOLA map viewer which ensured compatibility with geospatial standards such as Web Map Service (WMS) and Web Feature Service (WFS). Using GeoTools ensures SOLA will remain capable of supporting geospatial standards and allows GeoServer to be an optional component of the SOLA architecture.

Package names for GeoTools UI stem from `org.sola.clients.geotools`. Package names for SOLA GIS stem from `org.sola.clients.desktop.gis`.

6.4.1 UI

The UI packages include the Map class which extends the GeoTools `JMapPane`¹⁹ class, the main Swing GUI component of GeoTools, and the Utility, Messaging and Control Bundle classes.

The Control Bundles are GUI components that encapsulate the map control, map tools and map actions used for a specific instance of the map viewer. They also allow configuration of the data source to use and layers to display in the map viewer. The SOLA Controls Bundle includes Zoom and Pan tools as well as the Information Tool and is the default map viewer for SOLA. The Locate Application Controls Bundle further extends the SOLA Controls Bundle to include the Locate Application tool which allows users to identify the approximate location of a new Application during the lodgement process.

6.4.2 Map Tools

Map Tools require the user to interact directly with the map control using their mouse. This could include clicking on a point, selecting an area on the map control using a selection rectangle or using a sequence of mouse clicks to draw or edit a feature. The Map Tools currently available in SOLA include Pan, Zoom In, Information and Locate Application.

6.4.3 Map Actions

Map actions are triggered by the user selecting a button, toolbar item or menu option and typically result in changes to the map control, but they do not require the user to interact directly with the map viewer. The Map Actions currently available in SOLA include Zoom Out, Zoom to Full Extent and Locate Application Remove.

6.4.4 Layers

A layer represents a distinct collection of geospatial features for display. Examples include parcels, roads, road centre-lines, survey marks, place names, etc. The map viewer will typically display a number of layers concurrently to provide the user with the necessary location context and each layer can have its own symbolization and styling to differentiate it from the geospatial features in other layers. SOLA uses Styled Layer Descriptors²⁰ (SLD) to describe the symbolization and styling of layer features.

The data used for layers can be obtained from different sources. SOLA is able to display data from WMS, Shape file and custom POJO objects. The SOLA Development Snapshot uses POJO Layers with data sourced from the SOLA database via the Spatial Web Service.

6.4.5 Data

The POJO Data Access feature of SOLA uses the Well Known Binary²¹ (WKB) format to transfer data to and from the map viewer via the Spatial Web Service. WKB is the geospatial

¹⁹ <http://docs.geotools.org/latest/userguide/unsupported/swing/jmappane.html>

²⁰ <http://www.opengeospatial.org/standards/sld>

²¹ http://en.wikipedia.org/wiki/Well_known_binary



data storage format used by PostGIS so using WKB removes translation between different geospatial formats when sourcing data from the SOLA Database.

6.4.6 GIS

The GIS package provides a Starter class to initialize the SOLA GIS components with the Spatial Web Service. It also implements a Messaging class that overrides the default Messaging class in the GeoTools UI package to integrate with the SOLA Message Utility for display of localized messages.

6.5 Services Layer

SOLA implements a web services based architecture that is consistent with the principles of Service Oriented Architecture (SOA)²². This includes eight SOAP based web services implemented using the Java JAX-WS technology backed by eleven Enterprise Java Beans (EJBs) hosted using Glassfish and Metro.

The SOLA Desktop can use the SOLA Web Services independently or combined in order to support land administration business processes. All of the services are designed to be interoperable and available for reuse by other systems. Of note, Metro provides support for WS-* standards and is interoperable with .NET Windows Communication Foundation (WCF) technology.

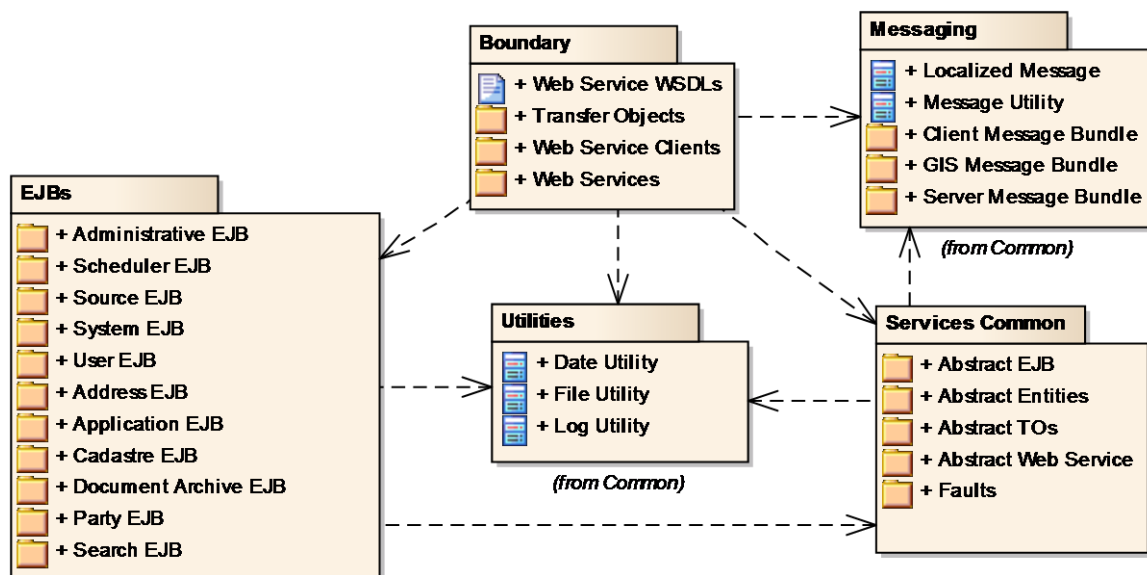


Figure 16 - Services Layer

Package names in Boundary stem from `org.sola.services`.

²² http://en.wikipedia.org/wiki/Service-oriented_architecture#Principles



6.6 Boundary

The Boundary contains packages that provide access to the SOLA Services Layer. Currently this includes eight SOAP based JAX-WS web services as well as the web service client classes. The Boundary package provides a logical location for any new SOAP web services as well as other forms of service interface such as RESTful web services or RSS services.

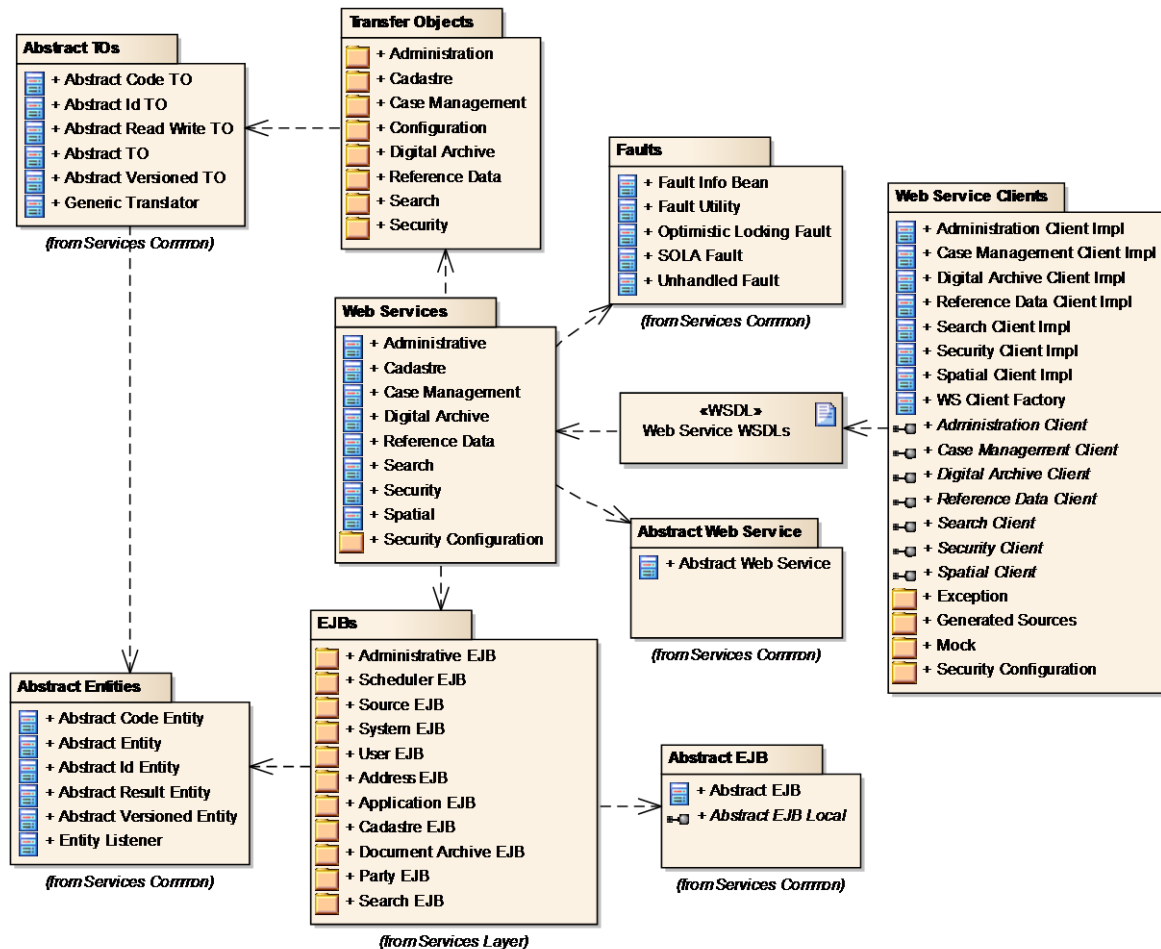


Figure 17 - Boundary

Package names in Boundary stem from `org.sola.services.boundary`.

6.6.1 Web Services

The web services current supported by SOLA are

- **Administrative Service** – Provides functionality to create, update and manage basic administrative units (a.k.a. BA Units) and related rights. This service is not part of the Development Snapshot but is currently in development.
- **Cadastre Service** – Provides functionality to maintain and manage cadastral processes and associated data.
- **Case Management Service** – Provides functionality to manage and track applications that initiate updates to administrative units.
- **Digital Archive Service** – Provides functionality to store and retrieve imaged documents.
- **Reference Data Service** – Provides access to system reference data (i.e. code lists).
- **Search Service** – Provides functionality to improve cross service search performance.



- Security Service – Provides functionality to create, update and maintain SOLA user details their security role mappings.
- Spatial Service – Provides direct read only access to spatial data used for navigating the SOLA GIS Map Viewer. This service is not configured with security as the overhead of message encryption negatively impacted the user experience when navigating (i.e. pan and zoom) the SOLA Map Viewer.

The web service classes contain no business logic. Their primary role is to present a facade²³ to the SOLA EJBs. They are also responsible for translation between Transfer Objects (TOs) and Entities and ensure a consistent approach to transaction and exception management is followed.

Security for the web services is configured using Web Service Interoperability Technologies (WSIT). This ensures the services are both secure and compatible with Microsoft .NET Windows Communication Foundation (WCF) technology. Refer to section 10 Security for more details on the SOLA Security Model.

Package names for Web Services stem from `org.sola.services.boundary.ws`

6.6.2 Transfer Objects

The Transfer Objects represent the data contracts for the web services and are an implementation of the Transfer Object pattern²⁴. They can be used to expose different views of the same underlying entity (e.g. PartyTO and PartySummaryTO). The TO's also provide a level of stability to the web service interfaces as changes to the underlying entities can be accommodated without affecting the TO's.

Package names for the Transfer Objects stem from `org.sola.services.boundary.transferobjects`.

Note that package-info classes are used with the transfer objects and with the web service classes to control the namespaces that get created for the generated client classes. These namespaces can also be used to support web service versioning where multiple deployed versions of the SOLA Web Services are required.

6.6.3 Web Service Clients

The Web Service Client classes are intended to be used by client java applications such as the SOLA Desktop to provide a consistent approach to accessing the SOLA Web Services. The Web Service Client classes act as proxies to the SOLA Web Services, establishing and managing connections to the services in the Services Layer as well as converting the faults thrown by the Web Services into more client friendly runtime (i.e. un-checked) exceptions.

This package also includes a set of mock service classes that can be used to test functionality in the Presentation Layer without requiring deployment of the SOLA Web Services.

Package names for Web Service Clients stem from `org.sola.services.boundary.wsclients`

6.6.4 Web Service WSDLs

Each web service has a WSDL xml file generated using the Java JDK wsgen²⁵ tool that describes the interface to the web service. The WSDLs can then be used as input to the

²³ http://en.wikipedia.org/wiki/Facade_pattern

²⁴ <http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html>

²⁵ <http://download.oracle.com/javase/6/docs/technotes/tools/share/wsgen.html>



wsimport²⁶ tool which generates the matching implementation classes for the web service interface in the Web Service Clients package.

6.7 EJBs

The EJBs encapsulate the main business logic for SOLA and have been implemented using the EJB 3.1 specification. This specification provides several improvements²⁷ over earlier EJB specifications and is intended to be light weight and functional.

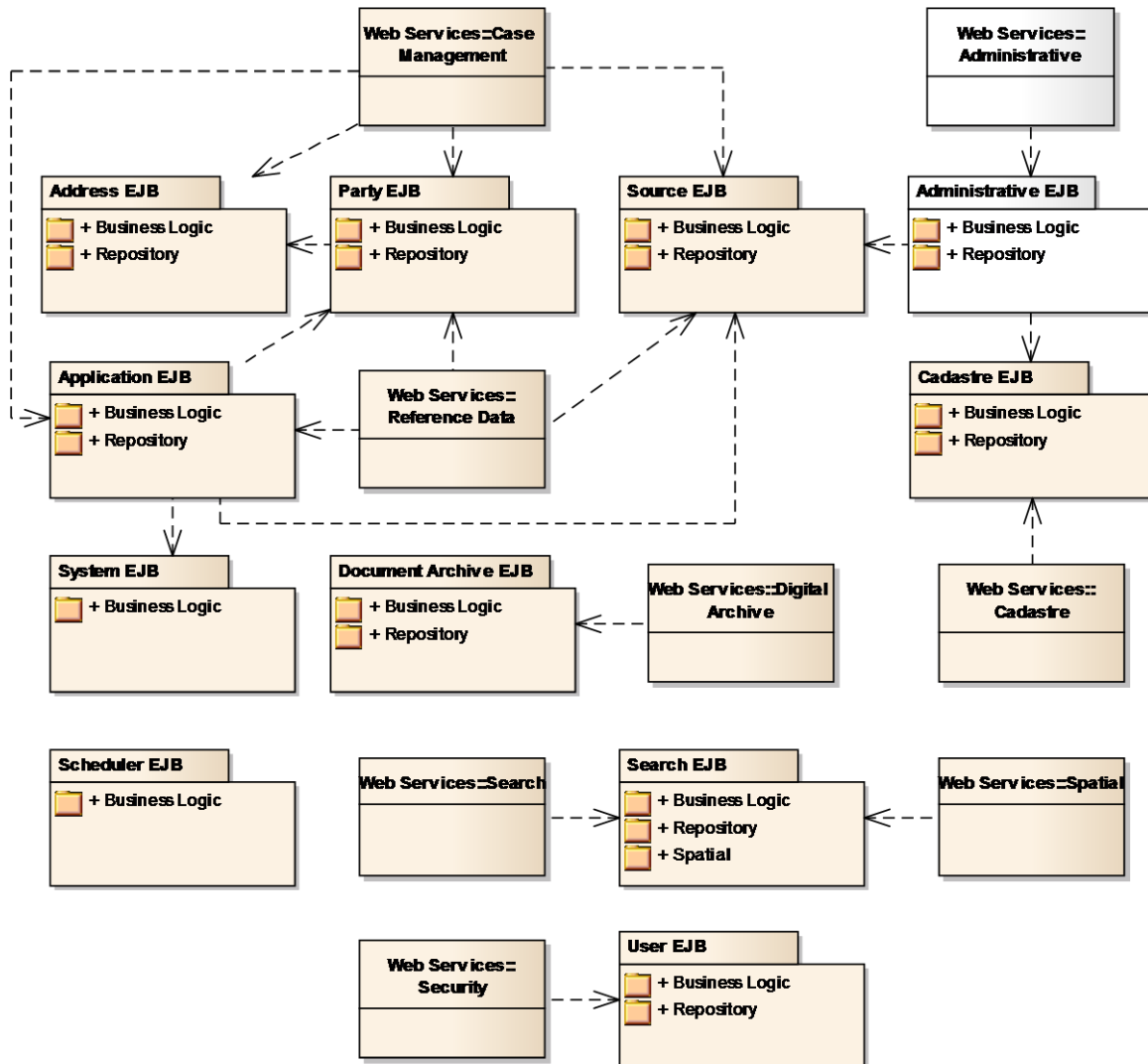


Figure 18 - EJBs

The web services leverage Dependency Injection to obtain references to the SOLA EJBs. This allows alternative SOLA EJB implementations to be deployed into the Services Host and used in preference to the default EJB implementations. For example, an alternative implementation for the Digital Archive EJB could be created that sources digital documents from an existing Digital Archive rather than (or as well as) the SOLA Document Schema.

²⁶ <http://download.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html>

²⁷ Convention over Configuration (CoC), Context and Dependency Injection (CDI), Portable Global JNDI Names, etc.



A standard package structure is used for all the SOLA EJB's with a Business Logic and optional Repository sub package. The Application EJB is discussed below and describes these sub packages in more detail.

Package names for the SOLA EJBs stem from `org.sola.services.ejbs`.

6.8 Application EJB

The Application EJB provides functionality to create, update and manage applications. Applications typically comprise of one or more customer services²⁸ the land administration agency will undertake in response to a customer request and a service may result in changes to property and/or rights.

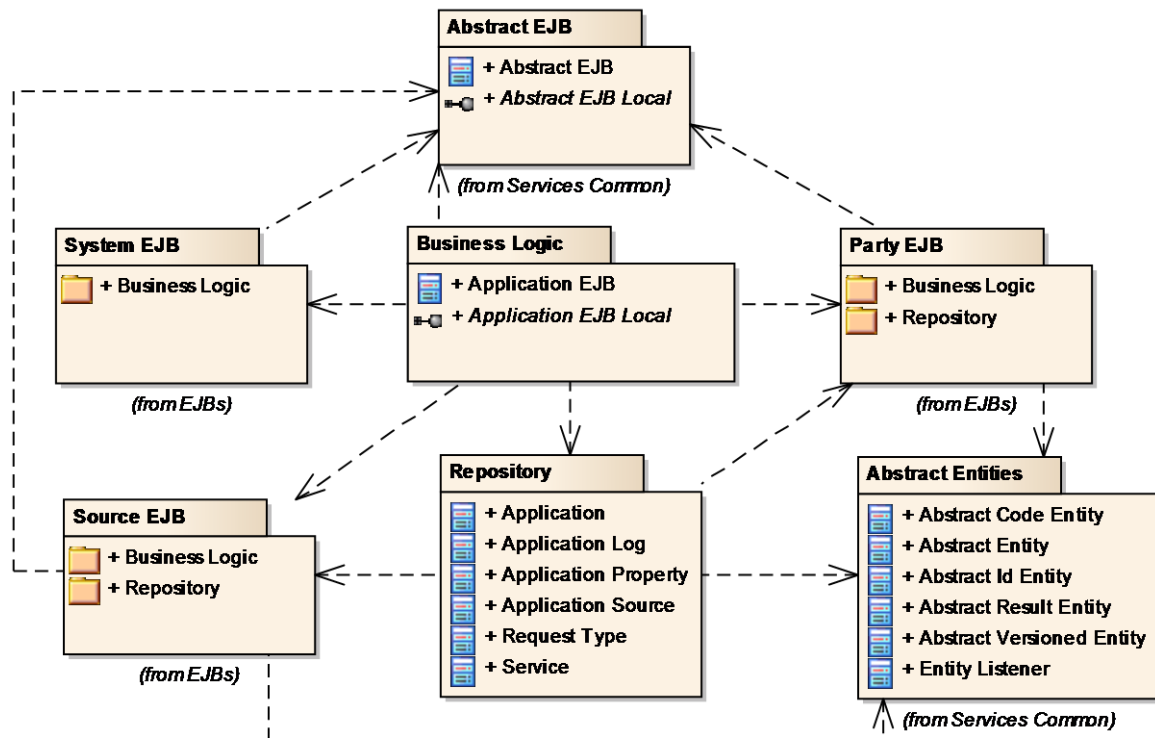


Figure 19 - Application EJB

Package names in the Application EJB stem from `org.sola.services.ejb.application`.

6.8.1 Business Logic

The Business Logic package contains the Stateless EJB class along with a Local interface. The EJB manipulates the entities from its repository layer to achieve business functionality. The Application EJB makes use of the Party and System EJB's to manage party and source (i.e. document) details. System parameters (e.g. tax rate) are obtained from System EJB.

6.8.2 Repository

SOLA uses Hibernate Core²⁹ and the Java Persistence API 2.0³⁰ (JPA 2) for Object Relational Mapping (ORM). The Repository package encapsulates the JPA Entity objects

²⁸ Not to be confused with web services provided in the SOLA Services Layer.

²⁹ <http://www.hibernate.org/>

³⁰ <http://www.objectdb.com/api/java/jpa>



used by the EJB to persist changes to the SOLA database. The JPA Entity Manager³¹ provided by Hibernate is supplied to the SOLA EJB's via dependency injection.

6.9 Services Common

The services common packages contains functionality that is shared between service layer packages.

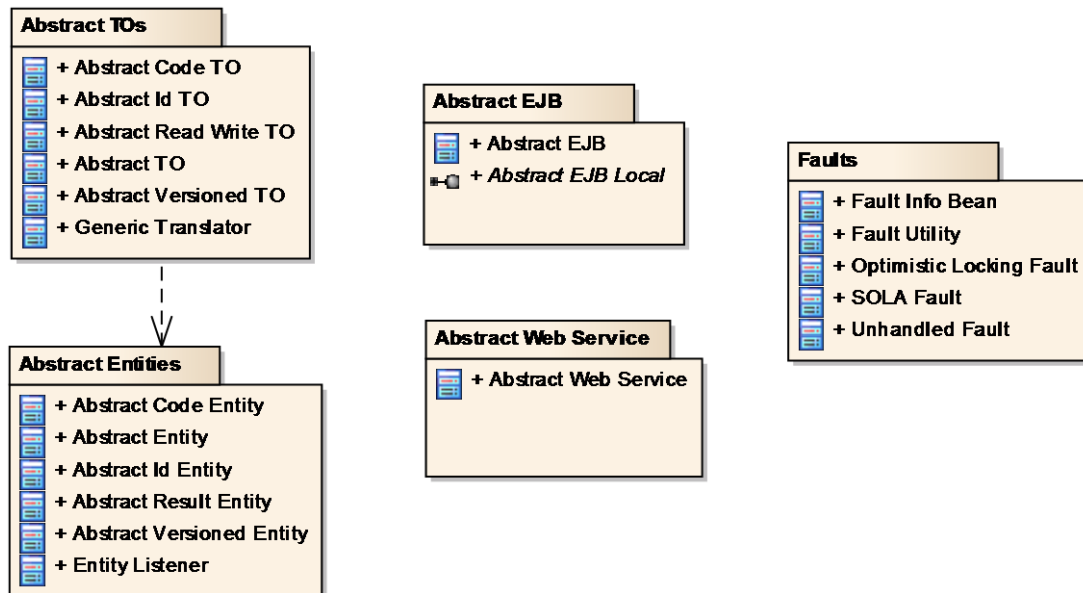


Figure 20 - Services Common

Package names in Services Common stem from `org.sola.services.common`.

6.9.1 Abstract TOs

This package contains abstract classes used by the Transfer Objects in the Web Services package as well as the Generic Translator. The Generic Translator is responsible for translating an entity object tree to or from a TO object tree. It uses reflection to achieve this task and relies on entities and TO's conforming to JavaBean conventions³². Using reflection avoids direct dependencies between the TO objects and the entities and means that the SOLA Web Services can be forward compatible³³ with updated versions of the SOLA EJB's so long as the EJB interfaces remain partially backwards compatible³⁴. For example, it may be desirable to update the SOLA EJB's for an existing SOLA implementation to the latest bug fixed versions without being forced to deploy the updated web service interfaces that accompany them.

The Generic Translator also plays an important role in merging data changes into entities. The use of Transfer Objects allows the SOLA Web Services to expose different views of the same underlying entity (e.g. `PartyTO` and `PartySummaryTO`), omitting attributes of the entity that are not relevant to the particular web service method. This complicates the process of saving data changes to existing entities because the JPA Merge method expects to merge two objects of the same type and it replaces all values on the object retrieved from the

³¹ <http://www.objectdb.com/api/java/jpa/EntityManager>

³² <http://en.wikipedia.org/wiki/JavaBean>

³³ http://en.wikipedia.org/wiki/Forward_compatibility

³⁴ Any previous method signatures on the EJB interfaces will need to be maintained, however any entities passed as parameters to or from those methods could have their attributes modified without breaking the web services.



database (i.e. the attached entity) with the values from the object being merged (i.e. the detached entity). The net result is that any attribute on the attached entity that was omitted from the Transfer Object is cleared (i.e. set to null or a default value) during the Merge – this is highly undesirable behaviour. To avoid this particular issue, the Generic Translator first retrieves the appropriate entity from the database and copies the data from the Transfer Object directly into the attached entity effectively replaying any data changes made by the client into the attached entity. The entity attributes omitted from the Transfer Object are not updated by this process and therefore maintain their existing values.

Note that the Generic Translator is currently implemented with custom code and the Dozer Bean Mapper library³⁵. The custom code will be replaced with Dozer.

6.9.2 *Abstract Entities*

This package contains abstract classes used by the Entities in the EJB Repositories as well as the Entity Listener. SOLA uses the Hibernate 3.5.0 JPA implementation (the version provided by Glassfish 3.1) unfortunately Hibernate does not appear to honour changes made to entities during the PreUpdate JPA lifecycle event. The Entity Listener is a Hibernate specific approach to implementing the equivalent functionality of JPA PreUpdate.

6.9.3 *Abstract EJB*

This package contains the Abstract EJB class providing common EJB functionality to all descendent EJB classes.

6.9.4 *Abstract Web Service*

This package contains the Abstract Web Service class providing common web service functionality to all descendent web service classes.

6.9.5 *Faults*

This package contains the SOAP fault exception classes used by the SOLA web services to pass exception information back to client applications. This includes a Fault Info Bean with message code, message parameters and a unique error number that can be used to locate a specific error in the SOLA Services Log. The three fault classes supported by SOLA are

- SOLA Fault – This fault is used to send SOLA specific error information back to client applications. This may include information about a failed validation and/or instructions the user can follow to correct or avoid the issue that has caused the error.
- Optimistic Locking Fault – This fault is raised by a SOLA Web Service to indicate that an optimistic locking exception has occurred while attempting to save data changes. By trapping this fault, the client application can implement custom handling of this error such as automatically retrieving the latest data and displaying that to the user for review.
- Unhandled Fault – This fault is raised for all unhandled exceptions raised by the SOLA Web Services. It is a catch all fault and simply states that an error has occurred. It is used to apply the Exception Shielding pattern³⁶ and prevents uncontrolled leakage of detailed application information outside of the web service boundary.

Note that specific fault classes have been preferred over one fault class with a fault type attribute to ensure compatibility with the .NET framework and the Microsoft Enterprise Library Exception Handling Application Block³⁷ which processes exceptions based on class type.

³⁵ <http://dozer.sourceforge.net/>

³⁶ <http://msdn.microsoft.com/en-us/library/aa480591.aspx>

³⁷ [http://msdn.microsoft.com/en-us/library/ff664698\(v=PandP.50\).aspx](http://msdn.microsoft.com/en-us/library/ff664698(v=PandP.50).aspx)



6.10 Common

The common packages contain functionality that can be shared with packages in other layers and/or provide functionality that could be readily used in other systems.

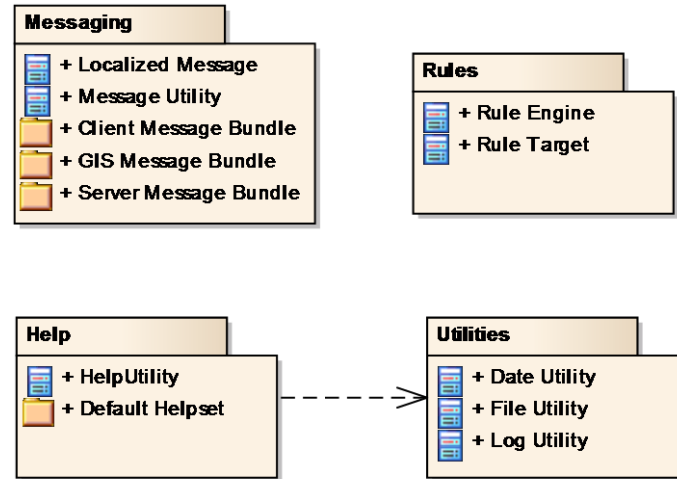


Figure 21 - Common

Package names in Common stem from `org.sola.common`.

6.10.1 Utilities

This package contains classes that provide common functionality to the Presentation and Service layers such as logging, date manipulation and related utility operations.

6.10.2 Messaging

The messaging package takes advantage of Java Resource Bundles³⁸ to provide message localization capability to SOLA. Every SOLA message is identified by a unique code which is used to retrieve localized message details from the Resource Bundle based on the Java locale settings.

6.10.3 Help

The Help package encapsulates the Java Help System³⁹ and can be configured with multiple localized helpsets providing SOLA with multi-lingual context sensitive help capability. As with the Messaging package, the helpset to display is based on the Java locale settings. Note that the Development Snapshot includes only one example helpset in English.

6.10.4 Rules

The Rules package encapsulates the Drools Expert rules engine and communication with the Drools Guvnor Business Rules Management System (BRMS). Note that the functionality provided by this package is under review and subject to refactoring.

6.11 Data Layer

The Data Layer persists SOLA data into a PostgreSQL database. The structure of the SOLA Database is based on the data storage requirements implied by the Land Administration Domain Model (LADM - draft ISO standard 19152) although extensions and adjustments have been included to support the function requirements of SOLA. The database contains multiple schemas with the data in each schema managed and maintained by a primary

³⁸ <http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/>

³⁹ <http://javahelp.java.net/>



SOLA EJB. The PostGIS Database provides support for storage and manipulation of spatial data.

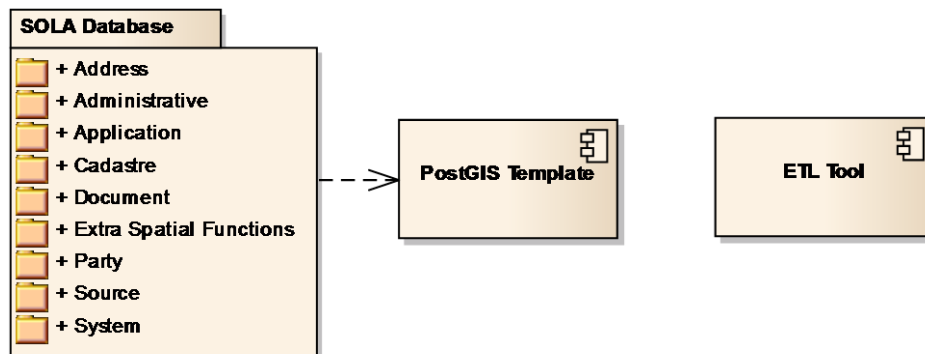


Figure 22 - Data Layer

The Extract Transform and Load (ETL) tool illustrated is an optional component of the Data Layer that could be used to perform data migration tasks if the land administration agency adopting SOLA has a requirement for data migration. Open source ETL tools supporting PostgreSQL include; Pentaho Kettle, GeoKettle, Talend Open Studio and Talend Spatial Data Integrator.

Refer to section 7.5 Data View or the FLOSS SOLA Data Dictionary for additional description of the SOLA Database.

6.12 External Systems

External Systems identifies the systems SOLA integrates with. The SOLA Desktop can direct print jobs through to print servers on the land administration agent's network. Email notifications will be sent via the land administration agent's email server. Scanners can be configured to place imaged documents on the network or local file location so they may be picked up and linked to the relevant application.

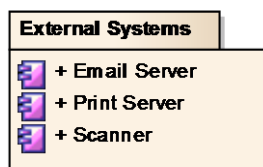


Figure 23 - External Systems



7. Deployment View

This section describes the physical configuration on which the software is deployed and run. It shows the physical (or virtual) nodes that execute or interface to the system and their interconnections. In addition to the illustrating the physical nodes used for deployment, the deployment diagrams that follow also illustrate the key communication paths between the various components along with the relevant communication protocols.

7.1 Deployment Models

SOLA is designed as a cross platform application capable of being deployed across any server and operating system configurations that support the Java Virtual Machine. SOLA must also support Centralized and Decentralized deployment (refer QL – 19, Hub-and-Spoke deployment model). This section illustrates the deployment models for

- SOLA Test
- SOLA Development Snapshot
- SOLA Developers Package

Further deployment models will be added to this section as they are established.

7.2 Deployment Artefacts

The deployment artefacts for SOLA are

- SOLA Services EAR
- SOLA Desktop Web Start
- SOLA Desktop
- SOLA Database

7.2.1 SOLA Services EAR

The SOLA Services Enterprise Archive (EAR) contains both the SOLA Web Services as well as the SOLA EJB's. Deploying the Web Services and EJB's in one EAR allows Local EJB interfaces to be used from the SOLA Web Services removing the need for cross boundary remote communication that would otherwise be required.

7.2.2 SOLA Desktop Web Start

The SOLA Desktop Web Start is a separate website package that contains a code signed version of the SOLA Desktop application along with a Java Network Launching Protocol (JNLP) descriptor file. The first time the user launches the SOLA Desktop from the web site, Java Web Start on the user's PC initiates download of the SOLA Desktop, installs it into the user's local application cache and runs the application. The user can then start the SOLA Desktop directly on their local PC using a desktop (or equivalent) shortcut referencing the cached version of the application. If a new version of the SOLA Desktop is deployed to the Application Server, the next time the user starts the application using their shortcut, the updated version will be automatically downloaded, cached and run.

7.2.3 SOLA Desktop

The Java Swing desktop application for SOLA. This is deployed to user PCs using Java Web Start technology as described above.

7.2.4 SOLA Database

The PostgreSQL database used to persist SOLA data.



7.3 SOLA Test

This diagram illustrates the deployment of SOLA components in the SOLA Test environment at FAO. Red-Hat Enterprise Linux v5.5 is used for the Application Server and DB Server.

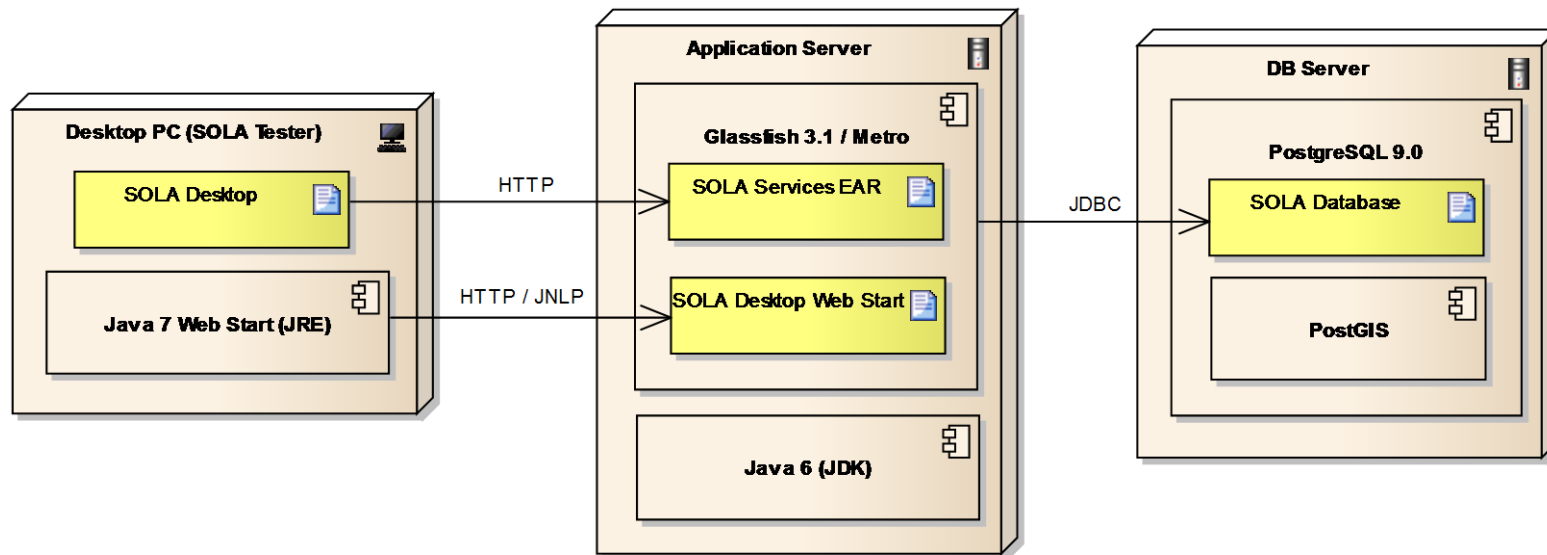


Figure 24 – SOLA Test Deployment Diagram



7.4 SOLA Development Snapshot

This diagram illustrates the deployment of SOLA components for the SOLA Development Snapshot. The SOLA Development Snapshot is a hosted environment that allows users to run the SOLA Desktop on their local PC using Java 7 Web Start Technology.

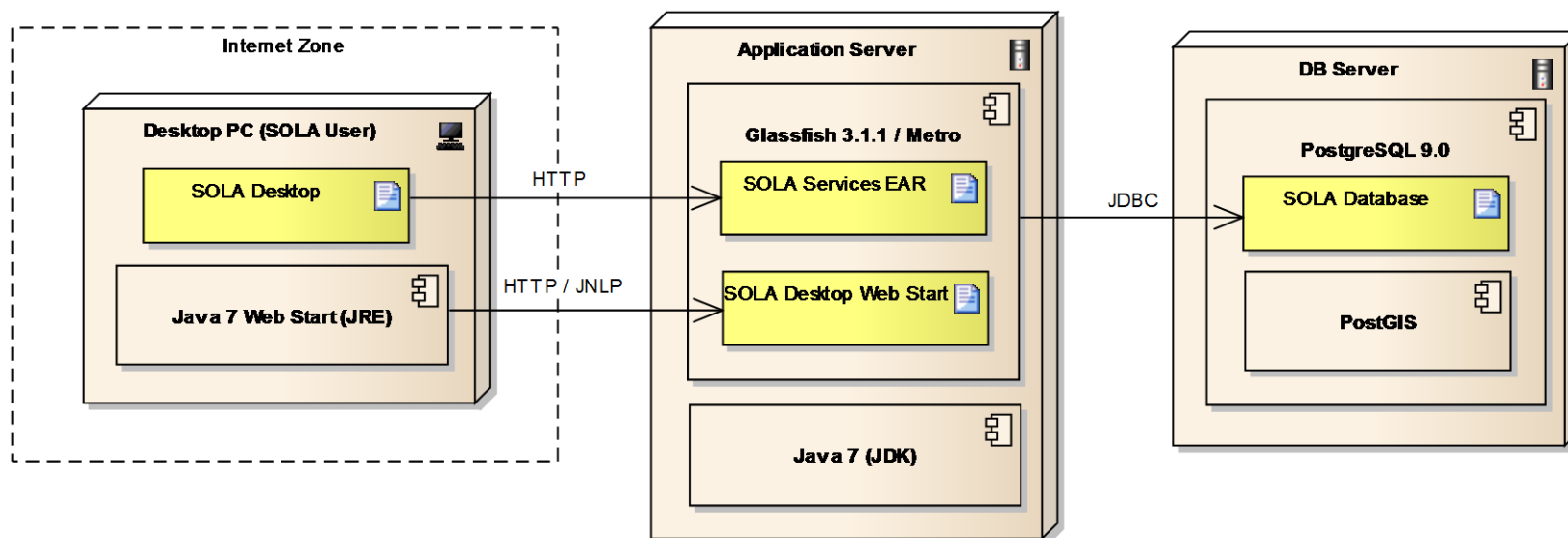


Figure 25 - SOLA Development Snapshot Deployment Diagram



7.5 SOLA Developers Package

This diagram illustrates the deployment of SOLA components in the SOLA Developers package that has been made available with the Development Snapshot. Netbeans 7.0.1 IDE is used for development of SOLA source code and can be used to compile and run the SOLA Desktop. The SOLA Database is packaged in an Ubuntu 11.04 Virtual Machine and is pre-loaded with the Development Snapshot test data.

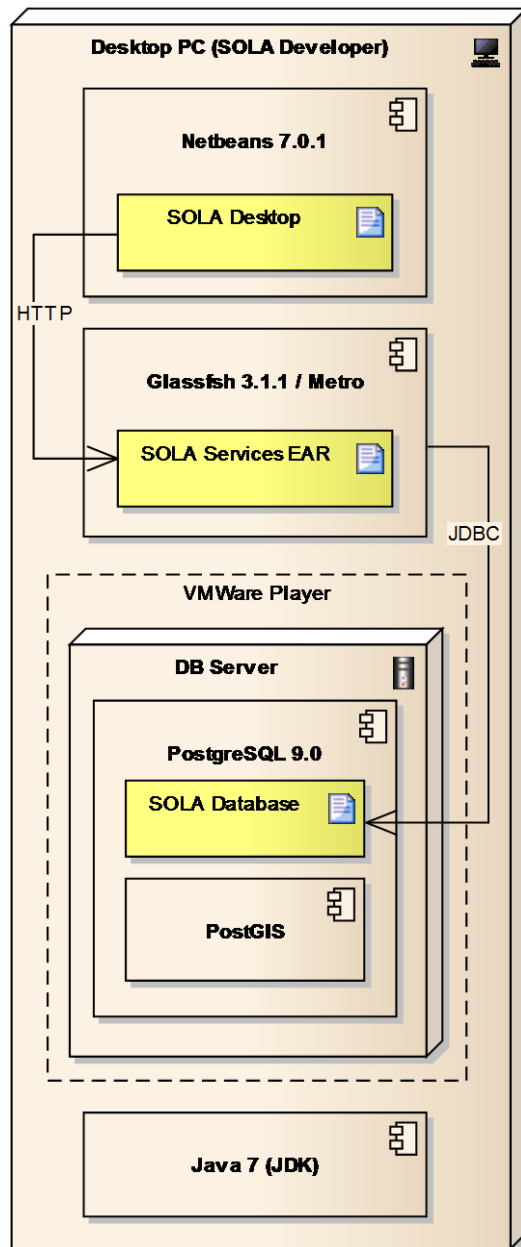


Figure 26 - SOLA Development Package Deployment Diagram



8. Data View

This section provides an overview of the architecturally significant persistent elements in the data model.

8.1 SOLA Data Model

The SOLA Data model uses the Land Administration Domain Model (LADM) to represent core land administration concepts and extends the LADM to satisfy the functional requirements of SOLA. The SOLA Data Model consists of one database containing a number schema with each schema having a primary SOLA EJB responsible for maintaining the data in that schema. Using multiple schemas improves overall data management and provides an additional level of data security control.

The schemas of the SOLA database are

- Address – Persists address details.
- Administrative – Persists data that identifies the legal rights of individuals and groups in relation to administrative units (e.g. parcels of land).
- Application – Persists data relevant for managing applications submitted to the land administration agency for the purpose of initiating changes to land rights and/or the cadastre.
- Cadastre – Persists data describing the cadastre managed by the land administration agency.
- Document – Persists copies of scanned documents as well as documents generated by SOLA for reporting and external correspondence.
- Extra Spatial Functions – Contains additional SOLA specific spatial functions.
- Party – Persists contact and related details for individuals, groups and organizations (i.e. parties).
- Source – Persists details about document types and their relationships to other entities in the SOLA Database.
- System – Persists data relevant to the administration and configurations of the SOLA application.

Note that further detailed information regarding the SOLA Database schema and tables can be obtained from the FLOSS SOLA Data Dictionary accompanying this Software Architecture Document.

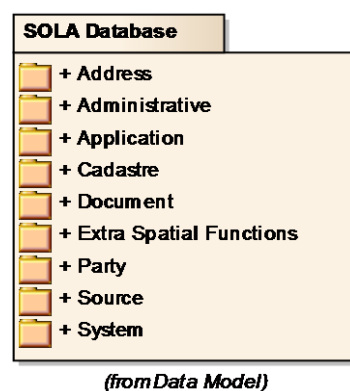


Figure 27 - SOLA Database Schemas



9. Size and Performance

This section discusses the size and performance characteristics of SOLA.

9.1 Use Case Points

Use Case Points (UCP) is an effective macro level estimating technique based on Use Case Models that can be applied during the inception and/or elaboration phases of a software development project to gauge the approximate size and effort required to develop the software. UCP accounts for both technical and environmental factors and is intended to represent the effort required for development, testing, business analysis and project management tasks. UCP does not account for effort related to transition activities such as data migration, deployment, configuration and general administration⁴⁰.

Three UCP estimates (likely, best and worst) were performed for SOLA based on the Use Case Model in section 4.1.2 of the SOLA Statement of Requirements v1.1 document. These estimates indicate the size of SOLA is in the range of 162 to 217 Use Case Points. Using 28 hours per Use Case Point, this equates to 4536 to 6076 person hours or approximately 27 to 37 person months effort⁴¹.

9.2 Database Size

The SOLA Desktop Snapshot Database is relatively small and contains example data only. It is approximately 6Mb in size. It contains (approx.)

- 65 Applications
- 70 Parties
- 60 Addresses
- 6,300 BA Units
- 8,500 Spatial Units

9.3 Code Metrics

SOLA uses Sonar⁴² to regularly review the SOLA code base and obtain code quality and sizing metrics. The Development Snapshot is approx. 30,000 lines of code. This figure includes generated code.

9.4 Performance

9.4.1 Performance Criteria

The performance criteria for SOLA are

- The system will be capable of supporting peak user volumes of up to 100 users (QL – 34).
- The system will process and respond to login requests in less than 5 seconds 90% of the time when subjected to peak user volumes (QL – 35).
- The system will process and respond to general transactions in less than 5 seconds 90% of the time when subjected to peak user volumes (QL – 36).
- The system will process and respond to process intensive transactions in less than 8 seconds 90% of the time when subjected to peak user volumes (QL – 37).
- The system will perform document generation in less than 30 seconds 90% of the time when subjected to peak user volumes (QL – 39).

⁴⁰ For more detail on UCP, refer to <http://www.methodsandtools.com/archive/archive.php?id=25>

⁴¹ Note that SOLA is a multi-phase project and the estimates listed above relate to the initial development effort only.

⁴² <http://www.sonarsource.org/>



10. Security

This section discusses the security characteristics of SOLA.

10.1 SOLA Security Model

The diagram below illustrates the security model that is used for the SOLA Development Snapshot. This security model leverages the features and capabilities of the JEE 6 Security Model, Glassfish and Metro.

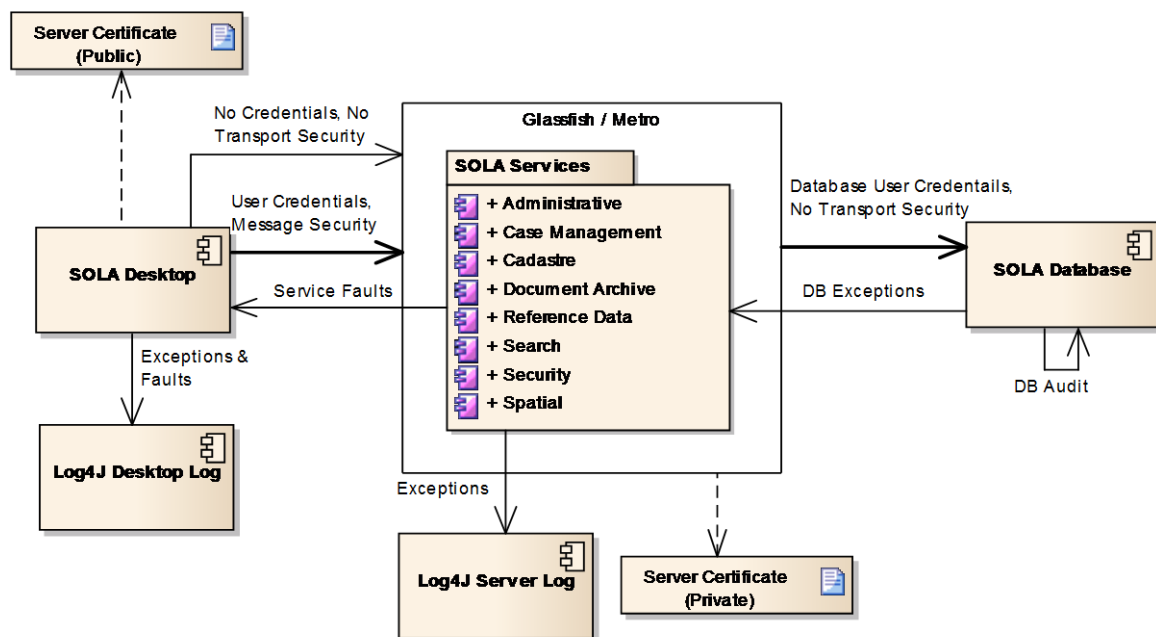


Figure 28 - SOLA Security Model

For each major communication flow (bold), the diagram identifies the identity that will be used for authentication and the security protocol being used. Supplementary communication flows for logging and exceptions details are also illustrated.

The sections below describe the SOLA Security Model in depth by addressing the following security aspects

- Authentication
- Authorisation
- Confidentiality and Integrity
- Auditing and Logging
- Exception Management
- Map Viewer Navigation

10.2 Authentication

Authentication allows you to confidently identify the clients of your application. These might be end users, other services, processes, or computers.

The JEE 6, Glassfish and Metro stack used by the SOLA offers a range of authentication options⁴³ that can be configured at the point of application deployment. The option that has been implemented for the SOLA Development Snapshot is Username Authentication with

⁴³ http://metro.java.net/guide/Security_Mechanisms.html



Symmetric Key⁴⁴. This option was selected for the SOLA Development Snapshot because it simplifies the overall configuration and deployment of the solution and removes dependencies on external systems such as an Enterprise Directory Service, allowing the Development Snapshot to be deployed via the internet. With this option, the username and password details are passed with each service call to authenticate the user. Authentication of user credentials is performed automatically by Glassfish using a JDBC Realm⁴⁵ configured to obtain user and role details from the SOLA Database.

To connect to the SOLA database, a Glassfish JDBC Connection Pool is used. This connection pool is configured with the credentials of a PostgreSQL database user account that has sufficient rights to read and update the SOLA database tables. Using a fixed identity for the database connection avoids the need to manage a large numbers of user accounts on the database (i.e. one for each SOLA user) and ensures SOLA can benefit from connection pooling. As the SOLA Security Model requires individual users to be identifiable in the data layer for audit purposes, username details are included in all database Create, Update and Delete (CRUD) operations.

10.3 Authorization

Authorization determines what system resources and operations can be accessed by the authenticated user. This allows authenticated users to be granted specific application and resource permissions.

Authorization is not yet implemented in the SOLA Development Snapshot, however future versions of SOLA will implement Declarative Authorization on every SOLA EJB method using the **@DeclareRoles** and **@RolesAllowed** metadata annotations⁴⁶. This will prevent a user from executing a particular function / method unless they have been granted the appropriate role. User role mapping information will be maintained in the SOLA Database and made available to Declarative Authorization via the Glassfish JDBC Realm used for authentication.

The Principle of Least Privilege⁴⁷ applies to the fixed identity used to connect to the SOLA database. Database role authorization will be used to limit the identity's access to only those database objects necessary to support SOLA application functionality.

10.4 Confidentiality and Integrity

Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users. Encryption is frequently used to enforce confidentiality where the privacy of messages is a key concern.

Integrity is the guarantee that data is protected from accidental or deliberate modification and is persisted in a consistent state. Integrity for data in transit is typically provided by using hashing techniques and message authentication codes. Maintaining data in a consistent state is typically provided by transacting data modification operations.

To ensure the confidentiality of service message details, Message Security⁴⁸ is used to encrypt the message contents and user credentials using Symmetric Key Cryptography⁴⁹. Encrypting the content of the service messages ensures they cannot be interpreted by

⁴⁴ http://metro.java.net/guide/Security_Mechanisms.html#ahicv

⁴⁵ <http://download.oracle.com/javaee/6/tutorial/doc/bnbxj.html#bnbxm>

⁴⁶ <http://download.oracle.com/javaee/6/tutorial/doc/bnbxj.html#bnbxu>

⁴⁷ http://en.wikipedia.org/wiki/Principle_of_least_privilege

⁴⁸ <http://devreminder.wordpress.com/security/transport-vs-message-security/>

⁴⁹ http://en.wikipedia.org/wiki/Symmetric-key_algorithm



network sniffers or similar hacking programs. It also protects against accidental or deliberate modification of the service messages while they are in transit.

Symmetric Key Cryptography requires a secret encryption key to be shared between the systems participating in the communication. In the case of SOLA, the secret key is generated by the SOLA Desktop as part of establishing a connection to a SOLA web service. The secret key is then encrypted using Glassfish's public key and the encrypted secret key is passed to Glassfish to decrypt (using its private key) and to initiate communication. The public key of Glassfish is packaged with the SOLA Desktop at compile time. To ensure the integrity of Glassfish's public key is maintained, the entire SOLA Desktop deployment package is code signed⁵⁰ to prevent unauthorized code changes.

Although transport security (i.e. SSL) can be configured for the connection between the SOLA Services and the SOLA Database, additional security has not been configured to simplify deployment and avoid any performance overhead. Note that the password of the fixed identity is passed in an encrypted form when establishing a connection to the SOLA Database.

Some SOLA application functions require the coordination of multiple SOLA EJB's, some or all of which may perform database updates. To ensure SOLA maintains data in a consistent state, it employs Java Transaction API (JTA) Transactions. Although not required for the SOLA Development Snapshot, JTA also supports Distributed Transactions through the Web Service Atomic Transactions⁵¹ provided by Metro.

Another important consideration for the data integrity of SOLA is the character set used by the SOLA Database. SOLA must be capable of accurately storing data from a wide range of languages and dialects. To achieve this, the SOLA database stores all data using the UNICODE (UTF-8) character set.

10.5 Auditing and Logging

Effective auditing and logging is the key to non repudiation and tracking of system events. Non repudiation guarantees that a user cannot deny performing an operation or initiating a transaction.

The SOLA Database uses triggers to capture details related to data changes (i.e. insert, update and delete). This ensures changes made via the SOLA Services are audited as well as changes made directly to the database. As a fixed identity is used for database authentication, by default all data changes captured by the triggers would be related to the identity. To ensure data changes for specific records can be directly related to a user, key database tables include a Change User field that is populated with the name of the user making the data change. In combination with the audit triggers, it is possible to identify the history of changes to a specific record along with the user that made each change.

10.6 Exception Management

Exception Management identifies how exceptions are handled within the system.

All SOLA Web Services apply the Exception Shielding Pattern⁵² to sanitise unsafe exceptions and replace them with safe exceptions (i.e. exceptions that do not reveal sensitive information about the inner workings of the service). To enforce this pattern, all

⁵⁰ http://en.wikipedia.org/wiki/Code_signing

⁵¹ http://metro.java.net/guide/Using_Web_Services_Atomic_Transactions.html

⁵² <http://msdn.microsoft.com/en-us/library/aa480591.aspx>



SOLA Web Services implement the **@WebFault** annotation to specify the faults that can be raised from the service.

The full details of all exceptions raised within the SOLA Web Services are logged to the Services Log using Log4J to allow further detailed assessment if required. To assist identification of a specific error in the log file, Unhandled Fault's raised by the SOLA Services also include a unique error number that can be included in the error message displayed to the user. The user can then reference this error number in any communications with the administrator or helpdesk to ensure fast and accurate identification of the specific error.

Database exceptions are returned directly to the calling service, sanitised and logged as described above. Unhandled exceptions that occur in the SOLA Desktop are displayed to the user and logged.

10.7 Map Viewer Navigation

Testing of the SOLA Map Viewer indicated the overhead of message encryption negatively impacted the user experience when navigating (i.e. pan and zoom) with the SOLA Map Viewer. To avoid this impact, the Spatial Service has been created to provide read only access to spatial data through an un-encrypted service connection. As this service is not subject to the full SOLA Security Model, care must be taken to ensure sensitive information is not exposed via this service. Updates of SOLA data via this service must also be avoided.



11. Appendix 1 – Definitions, Acronyms and Abbreviations

Term	Description
.NET	A software framework created by Microsoft that supports the development and execution of other software programs that target the framework.
ACID	Automaticity, Consistency, Isolation, Durability. The set of properties that guarantee database transactions are processed reliably.
CDI	Contexts and Dependency Injection. Defines a set of contextual services provided by JEE containers that make it easy for developers to use EJBs along with JavaServer Faces technology in web applications.
CoC	Convention over Configuration. A software configuration technique that avoids excessive configuration of a component by providing a default configuration (i.e. the convention) and allowing variances from the default configuration to be applied as necessary.
CRUD	Create, Read, Update, Delete. The basic functions used to persist any software entity to a database or other storage system.
CSV	Comma Separated Values. A basic text file format where data values are separated by a comma.
DI	Dependency Injection. See CDI.
EAR	Enterprise ARchive. A file format used by Java EE for packaging multiple modules into a single archive for deployment to a Java EE Application Server.
EJB	Enterprise JavaBean. A body of (java) code with fields and methods to implement modules of business logic. They can be deployed on any JEE compliant application server.
ETL	Extract, Transform and Load. Generally a tool that can be used to extract data from one data source (e.g. flat file, database, etc), transform and load the data into another data source (typically a database).
FAO	Food and Agriculture Organisation. Specialized agency of the United Nations.
GIS	Geographical Information System. A system designed to support display and manipulation of geographic data.
GML	Geography Markup Language. An XML grammar used to describe geographical features. GML is often used as an open interchange formation for geographic transactions on the Internet.
GUID	Global Unique IDentifier. A randomly generated unique reference number used as an identifier in computer software. Typically a 32 character hexadecimal string. The probability of the same GUID being generated twice is negligible.
Javadoc	A documentation generator that can be used to generate API documentation in HTML format from Java source code comments.
JavaMail API	Supports sending email notifications from JEE applications.
Java Swing	The primary Java GUI widget toolkit used for building graphical user interfaces for Java programs.
Java Web Start	A technology framework built into the Java platform that allows users to install and start application software directly from the internet using a web browser.
JAX-WS	Java API for XML Web Services. A Java API providing features to support development and implementation of XML based web services (a.k.a. Big web services).



Term	Description
JEE	Java Enterprise Edition. A collection of technologies and APIs for the Java platform designed to support Enterprise Applications which can generally be classed as large-scale, distributed, transactional and highly-available applications designed to support mission-critical business requirements. JEE is currently at version 6.
JNDI	Java Naming and Directory Interface. Provides a directory service interface that can be used by Java software clients to discover and lookup data and objects via name.
JNLP	Java Network Launching Protocol. A XML defined protocol that describes how to launch a Java Web Start application.
JPA	Java Persistence API. A JEE standards-based solution for persistence. Persistence uses an object/relational mapping approach to bridge the gap between an object-oriented model and a relational database.
JSF	JavaServer Faces. A JEE user interface framework for building web applications.
JTA	Java Transaction API. A JEE API that provides a standard interface for demarcating transactions including Distributed Transactions.
LandXML	An open standard XML format for transferring survey and civil engineering data between systems.
LADM	Land Administration Domain Model (Draft Information Standard as at 1 st Jan 2011 – ISO standard 19152).
Modified BSD	A Berkeley Software Distribution open source software license containing 3 primary clauses; two relating to the retention of the original copyright notice and one stating the copyright holder and contributor names cannot be used for endorsement of derived products. This license has been selected for the SOLA Development Snapshot.
ORM	Object Relational Mapping. A technique for converting data between incompatible type systems in databases and object oriented programming languages.
POJO	Plain Old Java Object. A basic Java class that does not rely on an underlying framework or object model beyond that offered by the core Java platform to provide its functionality.
RSS	Really Simple Syndication (a.k.a. RDF Site Summary). A web feed format that can be used to publish frequently updated works such as new headlines, blogs, regular reports, etc.
RESTful Web Services	Web services that are implemented using HTTP and the principles of REST (REpresentation State Transfer). RESTful web services using the HTTP methods to define their operations (e.g. POST, GET, PUT, DELETE).
SAD	Software Architecture Document. A document that provides an architectural description of a software system using a number of different architectural views to depict different aspects of the system.
SOA	Service Oriented Architecture. A software architecture where functionality is grouped around business processes and packaged as interoperable services.
SOLA	Solutions for Open Land Administration. Refer to the SOLA web site (http://www.flossola.org/)
SSL	Secure Sockets Layer. A cryptographic protocol for securing communications across the internet.



Term	Description
TDD	Test Driven Development. A development technique that reinforces software quality by requiring unit tests matching the specification to be in place early to ensure the code unit performs its necessary function.
TO	Transfer Object. A software pattern where serializable classes are used to group attribute values for transfer over a remote interface (a.k.a. Value Object).
UPS	Un-interruptible Power Supply. An electrical power unit (e.g. battery) that automatically provides emergency power when mains power fails. Typically used to ensure computer systems have sufficient time (e.g. 5-10 minutes) to power down gracefully.
UTF-8	UCS Transformation Format – 8 bit. A multi-byte character encoding for Unicode. UTF-8 can represent every character in the Unicode character set and is backwards compatible with ASCII.
WCF	Windows Communication Foundation. A framework component of .NET 3.0 that is used to build applications that inter-communicate.
WFS	Web Feature Service. An interface standard for obtaining or manipulating geospatial features over web based infrastructure.
WMS	Web Map Service. A standard protocol for serving georeferenced map images over web based infrastructure. The map images are generated by a map server using data from a GIS database.
WS-*	Web Service Specifications. A set of web service specifications designed and promoted by the Web Services Interoperability consortium.
WSDL	Web Service Description Language. An XML document that describes a web services interface.

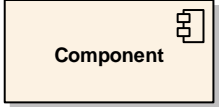
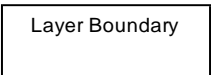
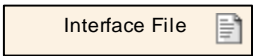
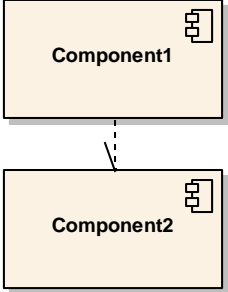


12. Appendix 2 – UML Notation

This appendix describes the UML Notation used through this document.

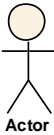
12.1 Component Model Notation

The table below provides a brief explanation of the UML notation used in Component Model diagrams.

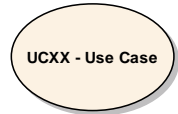
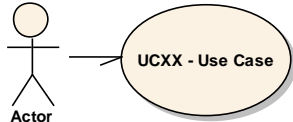
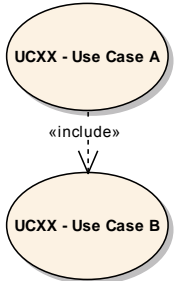
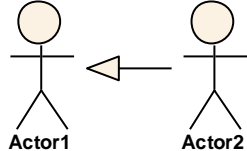
Element	Description	Notation
Component	A component is a modular part of the system, whose behaviour is defined by its interface. Components can represent a class, multiple classes a combination of other components or a combination of classes and other components.	
Layer Boundary	Identifies the layer boundary. Components contained within the boundary are part of the layer.	
Interface File	File containing data to be transferred between two or more systems.	
Dependency	Illustrates that Component1 is dependent on Component2 for the delivery of system functionality.	

12.2 Use Case Model Notation

The table below provides a brief explanation of the UML notation used in Use Case Model diagrams.

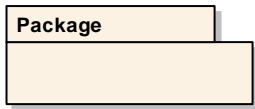
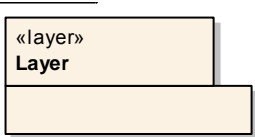


Element	Description	Notation
Actor	<p>Anything that exchanges information with the system. Two common types of actor are human actors and system actors.</p> <ul style="list-style-type: none">Human actors represent groups of users that have similar responsibilities and/or perform similar tasks with the system. Note that some users of the system may have responsibilities that span multiple human actors.System actors represent those systems providing supporting services and/or	








	data to the system.	
Use Case	A use case defines a set of actions a system performs that yields an observable result of value to a particular actor. The primary purpose of a use case is to capture the required system behaviour from the perspective of the end-user in achieving one or more desired goals	
Usage	An association between use case elements. It indicates one element uses the other in some way to achieve the goal of the use case. The arrow direction indicates which element initiates the usage.	
Usage (Include)	An include association indicates that the source use case includes the functionality of the target use case in order to achieve its goal. I.e. In all situations, the source use case represents some behaviour and/or functionality of the target use case.	
Generalisation	A generalisation is used to indicate inheritance, i.e. Actor 2 inherits the attributes and behaviour of Actor 1, in addition to their own.	

12.3 Logical View Notation

The table below provides a brief explanation of the UML notation and icons used in Logical View Model diagrams.

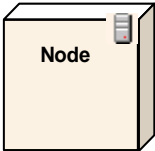

Element	Description	Notation
Package	A package is a namespace. It also represents the collection of elements that comprise the namespace. Packages can contain other packages and/or be encapsulated by other packages, subsystems or layers.	
Layer	A layer is a stereotype of package. Layers are used to decompose a system into smaller chunks so that it is easier to comprehend. Two common layering strategies are Responsibility Based and Reuse Based.	
Nested Package	A nested package illustrates a package, subsystem or layer nested within a parent package, subsystem or layer.	
Component	A component is a modular part of the system,	



	whose behaviour is defined by its interface. Components can represent a class, multiple classes a combination of other components or a combination of classes and other components.	
Database Object	A database object represents an element within a database. This includes tables, stored procedures, views, etc.	
Interface	An interface is a specification of behaviour that implementers agree to meet. It is often referred to as a contract. By implementing an interface, classes are guaranteed to support a required behaviour, allowing a system to treat otherwise non-related elements in the same way.	
Class	A class is a representation of object(s) that reflects their structure and behaviour within the system. It is a template from which actual running instances are created. Classes can have attributes (data) and methods (operations or behaviours). Classes can also inherit characteristics from parent classes.	
Form	A form is a stereotyped class representing a user interface screen or web form.	
GUI Element	A GUI widget control that can be placed on a window, form or other widget control.	

12.4 Deployment Model Notation

The table below provides a brief explanation of the UML notation used in Deployment Model diagrams.

Element	Description	Notation
Node	A node is a physical piece of equipment on which the system will be deployed - for example a server, disk array, or PC. A node usually hosts components and other executable pieces of code.	
Component	A component is a modular part of the system, whose behaviour is defined by its interface. Components can represent a class, multiple classes a combination of other components or a combination of classes and other components.	



Artifact	<p>An artifact is any physical piece of information used or produced by a system. Artifacts can have associated properties or operations, and can be instantiated or associated with other artifacts. Examples of artifacts include model files, source files, database tables, development deliverables or support documents.</p> <p>The artifacts in the following diagrams have been colour coded.</p> <ul style="list-style-type: none">• Pale artifacts represent existing artifacts within the target environment that will be used by SOLA• Light yellow artifacts are those created for SOLA and they must be deployed/updated as part of each system release• Light blue artifacts are automatically generated by components within the SOLA and they do not require deployment	<div>«artifact» Existing Artifact</div> <div>«artifact» Artifact to Deploy</div> <div>«artifact» Generated Artifact</div>
Zone	<p>The Zone the Node is deployed in. Note that it is expected that firewalls will exist between each zone.</p>	<div>Zone A</div>



13. Appendix 3 – Architectural Risks

The following table provides an overview and rating of the SOLA Architectural Risks. The fields in this table are described below

- Risk – The description of the architectural risk.
- Impact to Project – The impact the risk will have on the project (1 = Minor impact, 10 = Showstopper)
- Likelihood – The likelihood of the risk eventuating (1 = Low, 9⁵³ = Highly likely)
- Rating – The overall rating for the risk based on Impact * Likelihood (1 = Minor, 90 = Critical)
- Confidence – The confidence in resolving the issue should it eventuate (Low, Medium, High)
- Notes – Notes relevant to the risk

	Risk	Impact to Project	Likelihood	Rating	Confidence	Notes
1	Unable to configure the security of GeoServer to match the SOLA Security Model. Development Snapshot update – GeoServer is an optional component of SOLA mitigating this risk.	5	7	35	Low	Would need to identify the next best security options supported by GeoServer and use those.
2	The number of components required to implement SOLA significantly complicates installation, configuration and administration.	7	3	21	Medium	Ensure distinct components of the solution are kept to a minimum. Provide support material and automated installation packages. Use Maven packages to ensure portability of build environment.
3	The Open Source tools and components selected for SOLA lack the capabilities of equivalent commercial products limiting the features that can be implemented (e.g. GIS components).	6	3	18	High	Investigate alternative products.

⁵³ If likelihood is 10, then the item is an issue, not a risk.



4	SOLA is unable to meet its performance criteria	6	3	18	High	Performance testing to be undertaken and performance defects will be rectified as required.
5	Use of distributed transactions results in unreasonable complexity within the system and/or has significant performance impacts. Development Snapshot update – Local EJB's used for business logic encapsulation avoids distributed transactions mitigating this risk.	8	2	16	Low	
6	Performance of the SOLA Map Viewer is significantly impacted by large data volumes.	8	4	32	High	Testing to be undertaken to gauge impact.
7	Limitations of JPA API result in complex workarounds in repository implementation or workarounds in other architectural layers.	6	4	24	Medium	Can fall back to Hibernate which provides richer ORM functionality.



14. Appendix 4 – Additional Information

This appendix captures additional information regarding some of the architectural decisions that have been made as well as likely impacts to the architecture based on the expected future direction of the project.

14.1 Workflow Engine Evaluation

Significant consideration was given to incorporating a workflow engine into SOLA. A workflow engine provides flexible definition of workflow processes as well as supporting workflow functionality and would allow land administration agencies to configure workflows appropriate for their business practices. The cost of using a workflow engine is that it introduces significant configuration, integration and deployment complexity. The alternative, which has been selected to reduce overall system complexity, is to omit a workflow engine from SOLA and require land administration agencies to customise the SOLA Client application or develop their own client application to match their workflow and business practices.

Note that the services provided by SOLA will be capable of supporting a workflow engine. This will ensure SOLA can incorporate a workflow engine in future as well as allowing time for the current suite of open source workflow engines to further mature. It also means that agencies can choose to implement a commercial or open source workflow engine if they consider this is compatible with their needs. The workflow engine would primarily replace the SOLA Client application, but may also supplant some or all functions of the Case Management Service.

The following comparison table was used to assist the workflow engine evaluation.

	Workflow Engine	Customised Client Application
SOLA development effort	More effort required as the development team will need to learn how to install and configure the workflow engine for development purposes as well as document how it will be deployed and configured by land administration agencies.	Less effort required as development work can concentrate on development of the client application and the supporting services.
SOLA complexity	More complex. Configuration of workflows and integration with the workflow engine is likely to be non trivial.	Less complex. No workflows to configure and integration between the services is likely to be minimal.
Solution flexibility (runtime)	More flexible. Allows workflows that match the land administration agencies current practices to be implemented. Also allows these practices to be changed / evolve, although careful consideration will need to be given to any in process workflows when a workflow definition is modified.	Less flexible. Once the customised service is in place, changing the workflow will require updating and redeploying the software.



	Workflow Engine	Customised Client Application
Reporting and logging	Would include inbuilt reporting and logging features to capture metrics and statistics relating to timeliness and transaction throughput.	Reporting and logging to capture timeliness and transaction throughput statistics would need to be manually built in. Reporting could be customised to the agencies specific needs.
Testing (of SOLA)	Complex to test as will need to set up several different workflow scenarios to match those the platform might need to deal with.	Testing can focus on the supporting client application and services.
Agencies development / customisation effort	Difficult to predict. The complexity of the workflow engine and the number of customised workflow modules required could result in much the same effort that would otherwise be needed to build a custom service.	Is likely to require more effort than simply configuring the workflow engine, however this is difficult to predict.
Installation / setup in agency environment	Workflow engines can be complex to install and setup. Significant supporting documentation and walkthrough examples would be required to assist the agency in this task.	Relatively straight forward / well understood deployment process.
User Interface (UI)	Likely to be web based using some form of UI template mechanism so that the screen flows can be adequately managed by the workflow engine.	UI can be custom built to meet the requirements of the agency.
Match to agency requirements	Good. The workflow engine is very flexible but it might constraint the agency in some cases (e.g. options for implementing user interface, screen flows, etc).	Very good. The customised client application will be designed to match agency requirements.
Agency development team size	Small to moderate sized team depending on the complexity of the workflow processes to be modelled / implemented. May not need dedicated PM or BA resources and only require part time Test resources.	More likely to be a moderate sized team that includes PM, Dev, Test and BA resources.
Opportunity for technical knowledge transfer to agency team	Average. Agency team will be exposed to the workflow service and its configuration options, but may not need to dig through the code of the supporting services much and may not get a good opportunity to understand how these work.	Good. Agency team will learn the patterns used to implement the client application and supporting services when implementing the customisations.



	Workflow Engine	Customised Client Application
Agency upgrade path (to a workflow engine)	Workflow engine already in place so upgrading would not be required unless one engine is replaced with another.	The customised client application would be replaced with the equivalent client supported by the workflow engine. The workflow engine may also supplant some functions of the Case Management Service.
OS community incentives	Incentives to encourage new development by the OS community would be limited to workflow engine enhancements and development of new workflow modules and supporting services.	The opportunity to build / implement a core component for the platform would likely encourage OS community interest.
Likelihood of successful delivery	Less likely to result in successful delivery of the software as any issues with the workflow engine configuration or deployment will delay the project.	Using well understood design patterns and providing an example implementation of the client application will improve the chances of successful delivery.

14.2 Enterprise Service Bus

An Enterprise Service Bus (ESB) primarily addresses integration issues between systems and an ESB was not considered necessary for the initial development of SOLA. Should an ESB prove necessary when SOLA is deployed to a specific land administration agency, it will be possible to reconfigure and/or redeploy the SOLA services in an ESB (e.g. Glassfish ESB).