LIFE SWSS

"Smart Water Supply System"

LIFE14 ENV/PT/000508

Document Title:	SWSS architecture and requirements definition
Deliverable no:	B 6
Issue Date:	31/10/2016
Level of Dissemination (PU/CO):	CO
Author(s):	Pedro Galvão & Adélio Silva
Contributor(s)	

Table of Contents

Executive Summary

This document describes the design and technologies used in the SWSS platform. This software tool is responsible for gathering or indexing disperse sources of environmental data, and provide a single point of access. Over this catalog of data the platform will automate the execution of mathematical models or other data processing tools, report generation and alarms.

To allow multiple front-ends, the platform will expose its functionalities as services. This document also describes the architecture for a web portal where users can consume and configure the SWSS platform.

The document includes a description of the SWSS platform, SWSS web-portal and SWSS modelling services.

1 SWSS platform architecture

SWSS project is focused in providing an innovative management and decision support platform for water supply systems (WSS) under real working conditions. The SWSS Server goal is to provide this support through a set of services delivered by different front-ends (mobile, web and desktop).

Water supply systems (WSS) are large-scale systems that abstract, treat and transport water over vast geographical areas to consumers being crucial a safe and efficient operation of these systems. These systems leads to significant environmental impacts due to huge amount of energy consumed, greenhouse gas (GHG) emissions, and water leakages. Current control systems are mostly designed to deliver water not to provide water efficiently and so new approaches are required to put the focus on the efficiency. SWSS platform is a step forward in this direction enabling to build added value knowledge from the available data and, by this way, improving the operation by optimizing processes and reducing inefficiencies.

1.1 Assumptions

The SWSS server should connect to different front-ends (UIS). SWSS server will use a database backend to store persistent data. Hibernate will be used as a ORM to abstract way from database specific design. All server side code will be developed in .NET. Functionalities will be exposed as services (SOAP or REST). The web portal will follow a MVC architecture.

The used technologies require a windows compatible backend, but accommodate the possibility of using cloud infrastructure (AZURE). Access to the service should be made with modern browsers (no ie7 or bellow support will be available). The back end database will be SQL Server by default but PostgresSQL is also possible.

1.2 General description

SWSS platform is divided into server and client. The server is a collection of cloud services to manage environmental data. The client is a web portal where users can configure and consume these services. **Figure 1** describes the general actions for the platform.

Most actions are performed without human interaction by the "Automated Server" in regular intervals. These actions can also be triggered by users with sufficient privileges. Simple users can only configure how to consume information.

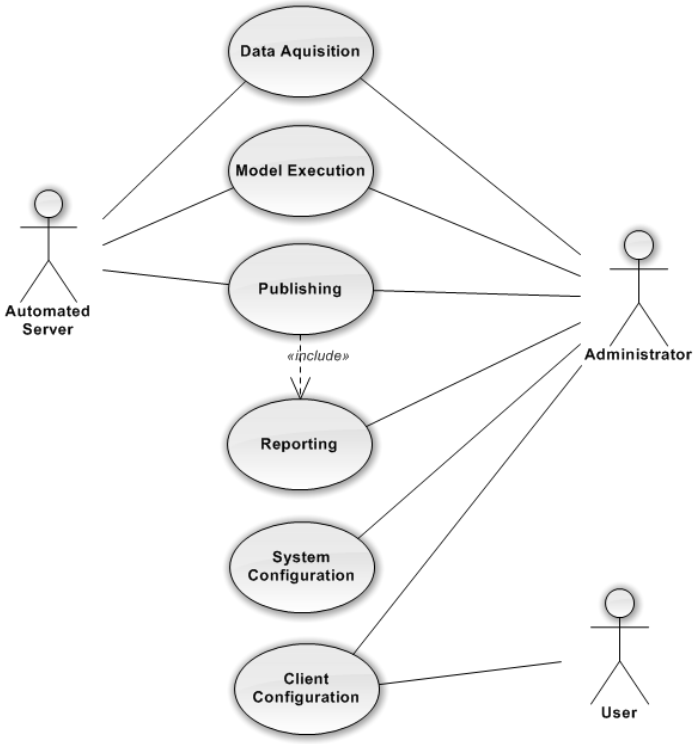

Figure 1 - SWSS platform use case

2 SWSS Server

SWSS Server is divided into 5 main modules:

- Data Storage and indexing
- Data Acquisition
- Model Execution
- Reporting
- Publishing

2.1 Data Storage and indexing

SWSS server needs to store or index data produced either by the Data Acquisition components or by Model Executions. Two main types of data are stored:

- Time series: data in a single point for a single property along time. For example data from a temperature sensor on a weather station
- Grid data: data on a structure or unstructured grid (1D, 2D or 3D) that varies in time. For example data from radar images or model results.
- Images: Images can be generated form GridData internally by the platform or from external sources

Time Series will be stored in a database. Grid and images will store metadata in the database and file on disk.

All Data is only stored for a limited interval in time. Automated backups/purges will be implemented.

2.1.1 Metadata definitions

The information configured in the system follows several metadata definitions in order to have a standard database. The metadata is important to resume the data

characteristics and group it in relevant information (e.g., the provider, the parameter, the geotemporal characteristics, the access and uses permissions - top of the Figure 2). For every data source in the system it is mandatory to fill its metadata.

In order to reduce the data processing and storage, systems should share the common data sources (grey pages on the flow of the Figure 2). To accomplish this the systems will communicate through metadata catalogues that will be stored in the Catalogue Server.

Therefore, the Catalogue Server is composed by a set of metadata catalogues from each system (the black box on the Figure 2). Each metadata catalogue contains the following information, which allow answering three questions:

1. The related system identification: Where is it from?
2. The system identification's that can access to these data sources: Who was access?
3. The list of the data sources metadata that this system want to share: What contains?

The metadata catalogue is created by every system administrator and published in the Catalogue Server. Because there are some specifics sources that are private (information cannot be shared with other forecast systems), this method protects this data sources.

The consumption of this catalogues is done by the system administrator and he can imports the available metadata catalogues. Focused in the flow of the Figure 2, the administrator can import the available catalogue for his system (in the flow, the administrator of the system E have access to the Catalogues A, B and C and he only want the Catalogue A and C) or add individually data sources from the metadata catalogues (represented by the green pages in the flow). Nevertheless new data sources can also be add in the system by completing a standard form.

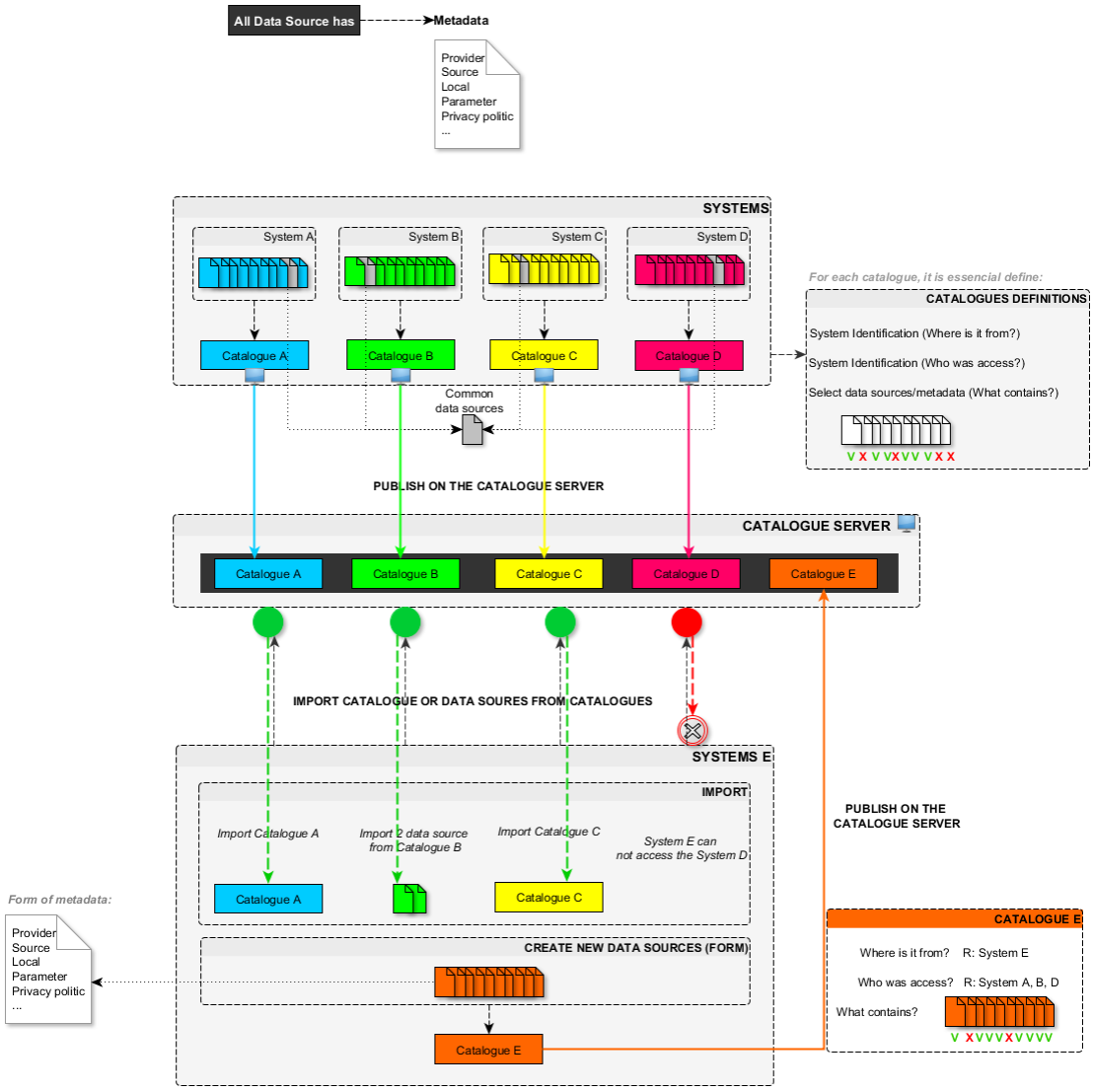

Figure 2: Scheme data sources in the forecast systems. Each system have a catalogue that it is publish in the Catalogue Server. A new system can import existents catalogues

2.2 Data Acquisition

Data is available in multiple formats (egg: asccii, xls, etc.. for Time Series or HDF5, NETCDF, GRIB for Grid Data). Multiple protocols can serve data, egg: FTP, http, OPEN DAP, Sensor Observation Service (SOS). Some types of data should be downloaded by the platform while others should only be indexed.

The data acquisition module is responsible for:

- Downloading or rebuilding indexes to known data sources
- Transform between known formats and the SWSS platform uses for storage
- Indexing the new information in the SWSS platform data store

All these processes should occur in pre-defined intervals or on-demand.

Each data acquisition cycle is encapsulated in a "DataDownload Job". This job is described by the following business diagram:

Figure 3: Download job workflow

Because the number and format of files is unknown the "Create Download List" and "Extract and convert downloaded values" will be abstracted by three interfaces IDownloadEnumerator and ITimeSeriesConverter and IGridFilesConverter. Each unique download source can use any combination of these interfaces to create a successful download. Pause and resume will not be supported in the initial version.

Figure 4: SWSS download interfaces

2.3 Models Execution

SWSS platform is able to execute mathematical models on user demand or pre-defined intervals. The work cycle for a model execution is assumed as:

- Data Preparation or input
    - Boundary conditions
    - Initial conditions
- Model Executions
- Data extraction

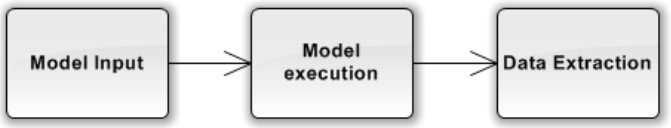

Figure 5: Model execution workflow

The data preparation step will create the necessary boundary and initial condition files. Boundary conditions will force the model execution throughout the duration of the model run. Initial conditions set the start values for model properties.

Sometimes initial conditions are obtained from the results of previous executions. This will be referred to as "Hotstart", in opposed do "Coldstart", when initial conditions are set from other sources or default values.

This work cycle is controlled by a "Model Job". The exact implementation of each of the steps in the work cycle is abstracted by two interfaces:

- IModelHandler interface will:
    - Create input files compatible with the respective model from the formats know by the SWSS Server
    - Create files compatible with SWSS server from model result files
    - Change model input files (eg: model start date and end date, Hotstart, etc…)

- IModelEngine interface will:
    - Execute the model (batch files, windows processes, etc)
    - Check if model execution was successful or not
    - Manage model Log Errors.

Each model implemented in the SWSS platform must implement these two interfaces. This will allow the platform to be expanded to other models in the future.

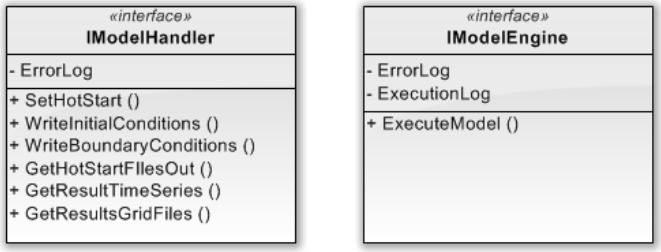

Figure 6: Model job interfaces

2.4 Reporting

Reports created by the SWSS platform can contain Time Series and images. Images can be created from grid files or Time Series indexed in the database, creating charts or maps. Reports can also contain raw Time Series values in the form of tables.

Reports can be created in office compatible formats (Open XML), pdf, xml or simple ASCII. Other formats can be incorporated in the future since the design general base structure is flexibly based on interfaces. A report is described by a report class:

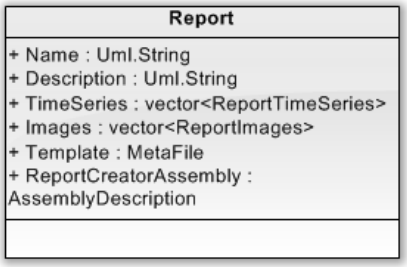

Figure 7: Report class

The report class contains the assembly that will use to create the report ("ReportCreatorAssembly", the platform can expand in the future to different report formats). "ReportTimeSeries" and "ReportImages" are classes containing the elements that can be placed in the report. Each contains a "ReportMarker" identifier so the report creator can correctly position it.

The class also contains a template file that is used by the report creator assembly. Report creation is triggered either by a Publishing job, Model Job or on user demand.

2.5 Publishing

Publishing is used to disseminate reports or alerts to a list of users. Publishing can occur by:

- E-mail
- SMS
- Http
- FTP
- Local file share

Publishing is represented by the Publication class:

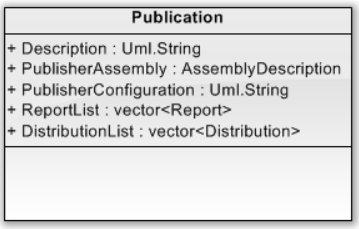

Figure 8: Publication class

The publisher assembly is responsible the dissemination protocol (sms, email, etc…). Configuration for each protocol is serialized in xml in the "PublisherConfiguration" property. The "DistributionList" property groups the target users, "ReportList" the reports to disseminate.

3 SWSS web portal

The SWSS web portal acts as a front end to the SWSS platform. Advanced users can manage the server creating new reports, data sources or automating new mathematical models. Simpler users can customize the information that is presented to them.

The web portal is the same for all areas of interest of the SWSS project. To avoid information overload the adaptive portlet design pattern will be used. Users can customize how to consume the information available by composing dashboards with relevant portlets. This design pattern is used in portal like Sharepoint.

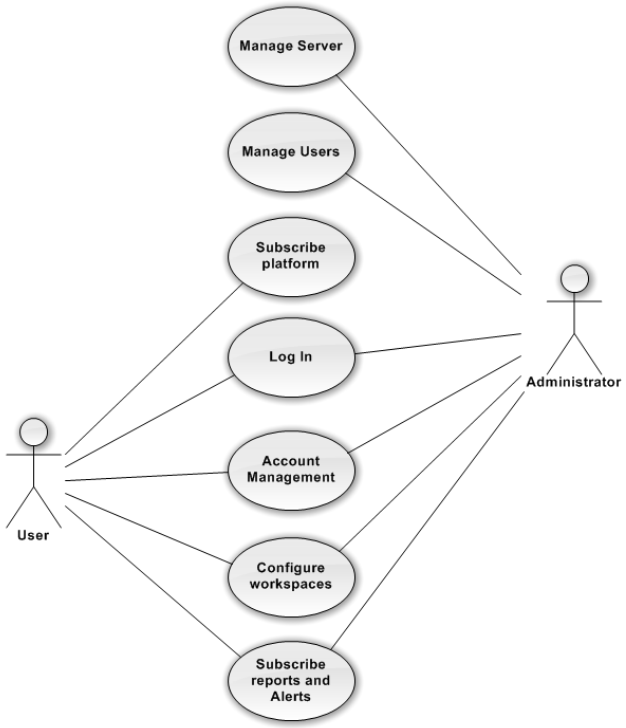

Figure 9: SWSS web portal

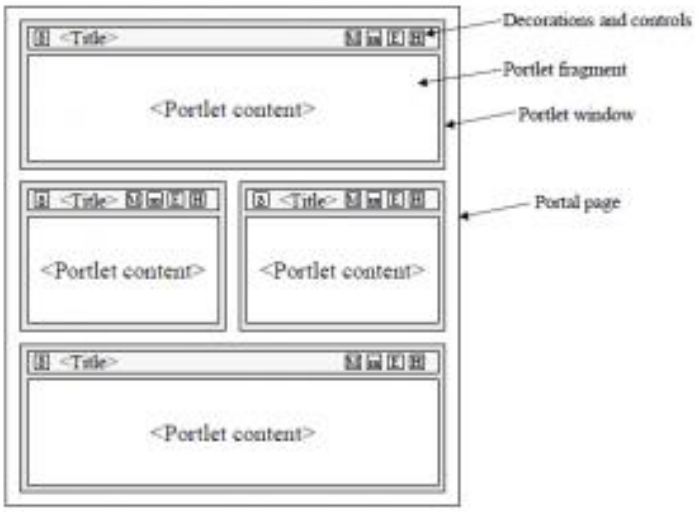

Figure 10: Adaptive portlet design pattern

The following Portlets are scheduled for development:

- Maps – allow users to visualize information in the form of maps.
- Line Charts – visualize time series as bar or line charts
- Table – Visualize time series as a table of values
- Image Viewer – Explore indexed images
- Report Indexer – list user favorite reports
- Execution log – check for automated job executions

Another advantage of this design pattern is that new portlets can be added to the platform at any time. The cost of expansion is only related to the implementation of the new control. If necessary this list of portlets can be reviewed and expanded depending on user feedback.

Communications between SWSS Server and portlets is done by REST services.

3.1 Design Strategies

The two main design strategies used in the SWSS platform are: Service oriented architecture and Inversion of control. An adaptive portlet design pattern will be used for the web portal.

3.1.1 Service oriented Architecture

SOA can be seen in a continuum, from older concepts of distributed computing (Bell, 2008) and modular programming, through SOA, and on to current practices of mashups, SaaS, and cloud computing (which some see as the offspring of SOA) (Watson, 2009).

Service-orientation requires loose coupling of services with operating systems, and other technologies that underlie applications. SOA separates functions into distinct units, or services, (Bell, 2008) which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.

This is particularly useful for the SWSS server. Each component is exposed has a service (Data Storage and indexing, Data Acquisition, Model Execution, Reporting, Publishing).

This allows for a greater modularity and improves scalability in a cloud environment. Resource intensive services like model execution or data acquisition can be spread along multiple servers. Multiple front-ends can consume data produced or indexed by the server by referring to the appropriate service. The services innards themselves can be replaced as long as they adhere to the predefined contracts of interaction exposed by their interfaces.

3.1.2 Inversion of control

Inversion of Control (IoC) is an abstract principle describing an aspect of some software architecture designs in which the flow of control of a system is inverted in comparison to procedural programming (Caprio, 2005).

In traditional programming the flow of the business logic is controlled by a central piece of code, which calls reusable subroutines that perform specific functions. Using Inversion of Control this "central control" design principle is abandoned. The caller's code deals with the program's execution order, but the business knowledge is encapsulated by the called subroutines.

In practice, Inversion of Control is a style of software construction where reusable generic code controls the execution of problem-specific code. It carries the strong connotation that the reusable code and the problem-specific code are developed independently, which often results in a single integrated application. Inversion of Control as a design guideline serves the following purposes:

- There is a decoupling of the execution of a certain task from implementation.
- Every system can focus on what it is designed for.
- The systems make no assumptions about what other systems do or should do.
- Replacing systems will have no side effect on other systems.

Inversion of Control is sometimes facetiously referred to as the "Hollywood Principle: Don't call us, we'll call you", because implementations typically rely on callbacks.

Inversion of control provides extensibility to the SWSS platform. Well defined interfaces were designed for data acquisition, transformation, model execution, reporting etc. These tasks are encapsulated in jobs that manage the information flux, but the details of model execution, publishing, etc… are contained in the interface implementation.

3.1.3 Adaptive portlet design pattern

"Portlets are web components--like servlets--specifically designed to be aggregated in the context of a composite page. Usually, many portlets are invoked to in the single request of a portal page. Each portlet produces a fragment of markup that is combined with the markup of other portlets, all within the portal page markup." (from the Portlet Specification, JSR 168).

The Portlet specification defines a portlet as a "Java-technology-based web component, managed by a portlet container that processes requests and generates dynamic content." However this concept was adopted by other technologies.

Figure 11: IGoogle portal (portlet style)

The main parts of this design pattern are the portal and the portlets. Portal functionality can be divided into three main parts (Mahemoff, 2006):

- Portlet container: A portlet container is very similar to a servlet container, in that every portlet is deployed inside a portlet container that controls the life cycle of the portlet and provides it with necessary resources and information about its environment. A portlet container is responsible for initializing and destroying portlets and also for passing user requests to it and collecting responses.

- Content aggregator: As defined in the Portlet Specification, one of the main jobs of a portal is to aggregate content generated by various portlet applications.
- Common services: One of the main strengths of a portal server is the set of common services that it provides. Services are not part of the portlet specification, but commercial portal implementations provide a rich set of common services to distinguish themselves from their competitors. A few common services that you can hope to find in most implementations are:
    - Single sign on: Allows you to get access to all other applications once you log into the portal server, meaning you don't have to log into every application separately. For example, once I log in to my intranet site, I should get access to my mail application, IM messaging application, and other intranet applications, without having to log into each of these applications separately. A portal server provides a secured credentials store, so users can authenticate only once for all portlets.
    - Personalization: The basic implementation of personalization service allows a user to customize her page in two ways. First, the user can decide what colors she wants for title bars and what icons she wants for controls. Second, the user can decide which portlets she wants on her page.

Portlets are web components that are deployed inside of a container and generate dynamic content. Portlets can:

- Be managed by a specialized container.
- Generate dynamic content.
- Life cycle is managed by the container.
- Interact with web client via a request/response paradigm.

4 References

Sprint, Planning . (2009). Retrieved from Sprint Planning Rules.

Ambler, S. W. (2005). *The Elements of UML(TM) 2.0 Style.* Cambridge University Press.

Beck, K. e. (2001). *Manifesto for Agile Software Development.* Agile Alliance.

Bell, M. (2008). *Introduction to Service-Oriented Modeling. Service-Oriented Modeling: Service Analysis, Design, and Architecture.* Wiley & Sons. ISBN 978-0-470-14111-3.

Caprio, G. (2005, September). Dependency Injection. *MSDN Magazine*.

Layers, O. (n.d.). *OpenLayers: Free Maps for the Web*. Retrieved from http://openlayers.org/

Mahemoff, M. (2006). *Ajax Design Patterns.* O'Reilly.

NHForge. (n.d.). Retrieved from Nhibernate: http://nhforge.org/

OGC. (n.d.). *OpenGIS Web Map Service (WMS) Implementation Specification*. Retrieved from http://www.opengeospatial.org/standards/wms

Quart.Net. (n.d.). Retrieved from Quart.Net: http://quartznet.sourceforge.net/

Schwaber, K. (2004). *Agile Project Management with SCRUM.* Microsoft Press. ISBN 978-0-735-61993-7.

Subversion. (n.d.). Retrieved from http://subversion.apache.org/

Sutherland, J. (. (2004). *Agile Development: Lessons learned from the first Scrum.*

The Bug Genie. (n.d.). Retrieved from http://www.thebuggenie.com/

Watson, R. (2009). *burtongroup.com*. Retrieved from burtongroup.com: http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html