Paper for Consideration by the S-100WG Focus Group / Test Strategy Meeting

Proposed Changes to the S-100 Portrayal

Submitted by:	SPAWAR Atlantic
Executive Summary:	This paper proposes two changes to the S-100 portrayal to address noted deficiencies. A companion proof-of-concept implementation of the proposed changes will be made available on www.basecamp.com.
Related Documents:	S-100
Related Projects:	S-100 Test Bed Projects (Simple Viewer, Shore Based ECDIS, ECDIS)

Introduction / Background

Extension languages extend or modify the capabilities of a software application. When used in this manner the code written in the extension language is generally referred to as a "script". By writing scripts, the developer or end user can implement new or updated functionality without modifying the application.

The current S-100 portrayal uses XSLT as an extension language to externalize portrayal processing for applications which render S-100 data. The application provides an XSLT interpreter/processor, and the portrayal catalogue provides XSLT "scripts" and support files which translate S-100 feature instances into drawing instructions. The application then interprets the drawing instructions in order to render a portrayal. This process is shown in figure 1.

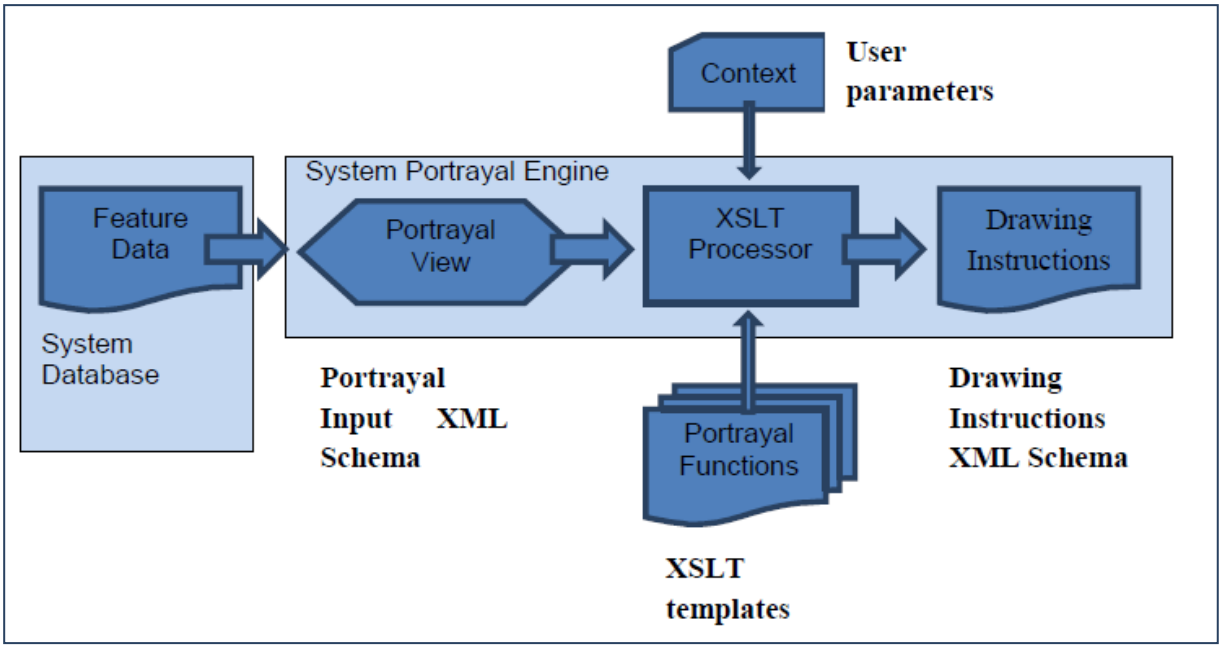

Figure 1 - Existing XSLT portrayal process

From our experience implementing the current portrayal model in an S-100 simple viewer, and based on preliminary recommendations for an Alerts and Indications model and a Product Interoperability model, we propose two changes to the S-100 portrayal model.

Analysis / Discussion

The changes proposed are to facilitate the development and implementation of Alerts and Indications, and Product Interoperability. These changes also enable simple implementation of the S-101 conditional symbology procedures (CSPs), and simplify product development for all S-100 based product types. The proposed changes are: augmenting or replacing the use of XSLT 1.0 as the S-100 portrayal extension language, and removal of the S-100 portrayal input schema.

The main issue with using XSLT 1.0 as the S-100 application extension language is that it is not well-suited to implementing complex state-dependent algorithms. Examples of such algorithms include the proposed S-100 Product Interoperability design, and the S-101 CSPs.

Level three and four of the proposed Interoperability Processing will be extremely difficult to implement using the current portrayal design. These levels require product portrayals to have visibility to one another, and to be able to modify each other's processing. The changes we propose facilitate implementation of the proposed Product Interoperability design.

The S-101 CSPs are described as state-machines, and some have proven resource intensive to implement. In particular, the safety contour generated by DEPARE03 has yet to be implemented correctly, and it has been proposed that the ECDIS should be responsible for implementing this CSP. We feel that this will lead to several undesirable results:

- CSPs implemented in the ECDIS will not be available for extension processing. This will affect further development of an Alerts and Indications model, and will likely affect development of the Product Interoperability model.
- The fact that a CSP could not be easily implemented via an XSLT script indicates that future products which require complex portrayals will likely also have to be implemented directly by the application.
- CSPs implemented within an application can only be updated by updating the application. The feature types the CSPs evaluate cannot be modified without potentially breaking the application.

For the reasons described, we propose augmenting or replacing XSLT with a different extension language. The new language should meet the following goals:

1. Require a minimum of changes to the current portrayal model.
2. Portrayal development for product creators, maintainers, and manufacturers should be simplified.
3. The development effort required to generate portrayal scripts should be reasonable.
4. Facilitate development of Alerts and Indications and Product Interoperability models within a reasonable timeframe.
5. Enable 100% updateable replacement for S-52 DAI files and CSPs.
6. Have permissive licensing.

Many extension languages are capable of meeting the stated goals. We are recommending Lua as the S-100 application extension language because:

- It was developed for use as an extension language, and is widely used for that purpose on desktop, mobile, and embedded platforms; all with varying hardware architectures and operating systems.
- It is written in pure ANSI C, compiles unmodified on all known platforms, and compiles cleanly as C++.
- It has a simple syntax yet is powerful, proven, and robust.
- It is easy to integrate into both new and existing applications, and is fully documented. Lua can integrate easily with applications written in C, C++, C#, Java, Python, Basic, etc.
- It has a permissive license (MIT).

We propose augmenting or replacing the process shown in figure 1 with the process shown in figure 2:

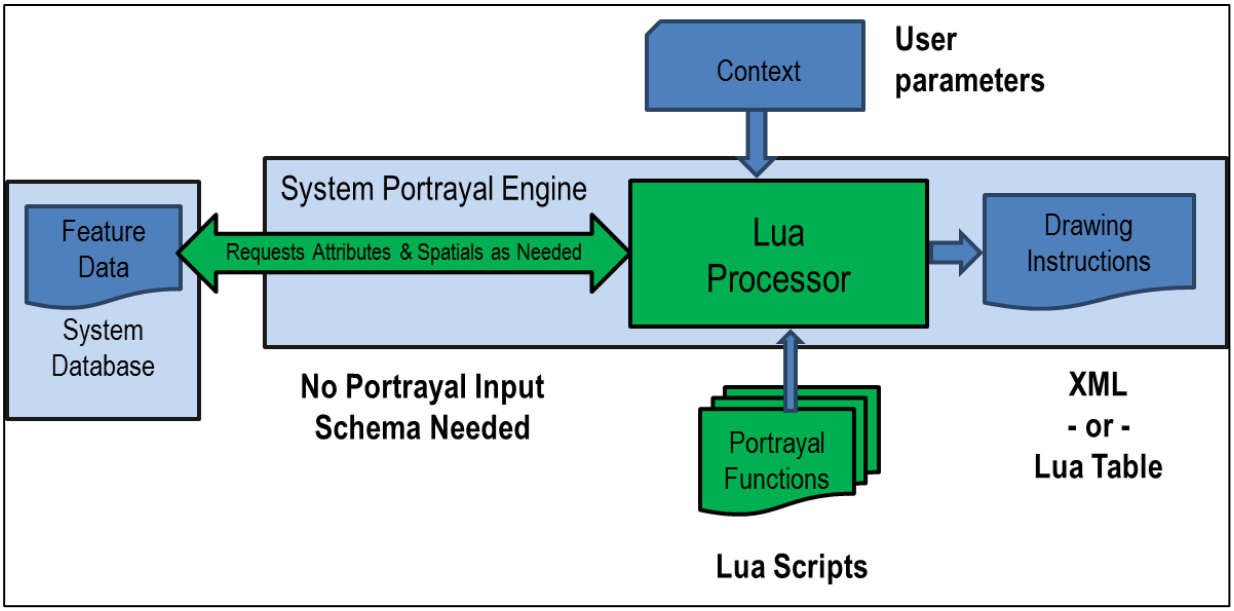

Figure 2 - Proposed Lua portrayal process

Green indicates proposed changes to the current portrayal. In the proposed Lua portrayal process all the support files are unchanged from the existing process, as shown in figure 3. This includes the symbols, colour profiles, line styles, area fills, fonts, etc. The drawing instructions can be output as an XML document conforming to the existing portrayal output schema. The primary change is that the portrayal functions (the scripts which reference the support files and generate the drawing instructions) are implemented in Lua vice XSLT. This change requires the application to implement a Lua interpreter as opposed to the current portrayals requirement to implement an XSLT 1.0 processor.

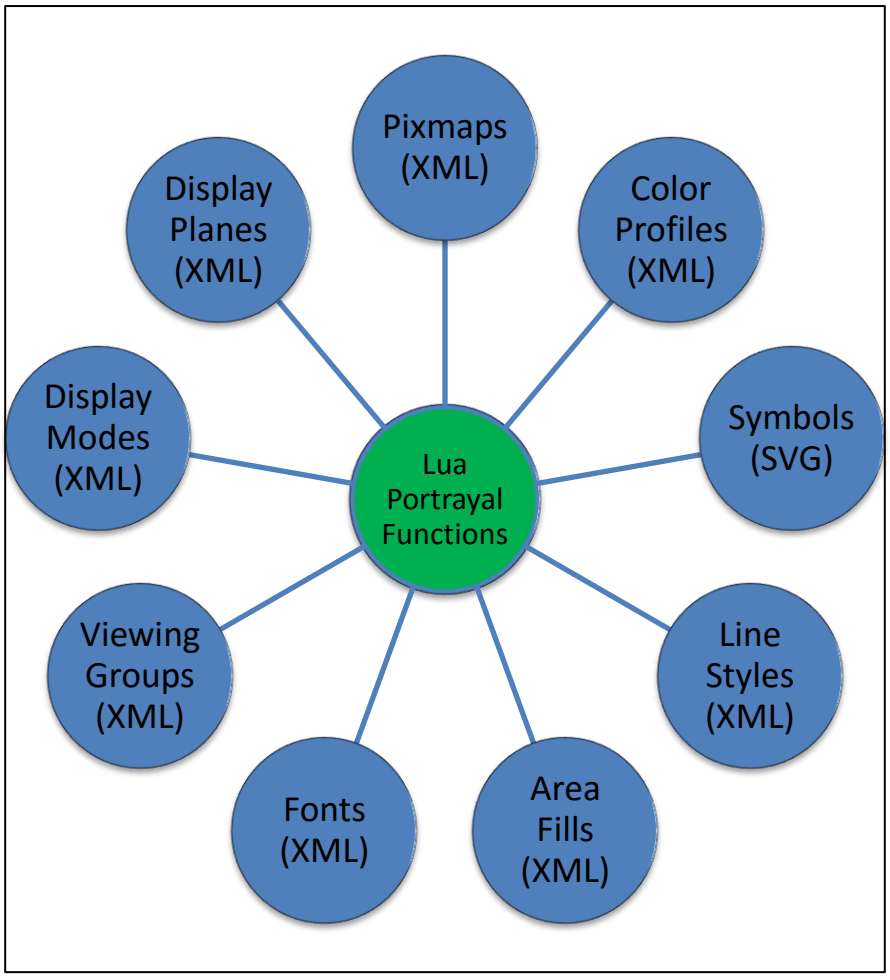

Figure 3 - Portrayal catalogue changes

The proposed portrayal process also does away with the existing requirement for a portrayal input schema. As noted in S-100 edition 2.0.0 Appendix 9-B-1, each product type is currently required to define a product input schema. An incomplete sample input schema for S-101 is included in S-100 edition 2.0.0 appendix 9-B-7. There are several drawbacks to requiring this input schema:

- It imposes additional complexity to the development of each S-100 product. This is exemplified by the fact that the required S-101 portrayal input schema remains incomplete. The sample S-101 portrayal input schema provided in Part 9 includes just four feature types.
- It creates a dependency between the portrayal catalogue and feature catalogue, making it difficult to maintain each independently.
- It requires applications to be able to coerce their SENC into the described structure, and to be able to adapt dynamically to any future changes in the products portrayal input schema. It is likely that changes to the portrayal input schema will break at least some applications.
- It makes it extremely difficult to develop applications which are capable of dynamically portraying new S-100 product types. New product types require a new portrayal input schema, likely requiring application code changes.

In contrast to XSLT, Lua can interact directly with the application, and therefore can request feature instance attribute and spatial elements from the application as they are needed during rules evaluation. This enables removal of the portrayal input schema, eliminating the disadvantages described above. In particular, an application should be able to portray new product types with no code changes required. In order to provide this functionality, the application must provide the following functions which will be called from the Lua portrayal scripts:

Application Provided Function	Purpose
DatasetGetFeatures(dataCoverageID)	Tells Lua the feature IDs contained in a dataCoverage
GetAttributeValue(featureID, attributeCode)	Tells Lua the value of a features attribute
GetSpatial(spatialID)	Tells Lua the value of the spatial
GetInformationAssociation(informationAssociationID)	Tells Lua the information association

Table 1 - Application Provided Functions

To generate the drawing instructions for a given feature instance, the application will provide the Lua portrayal with the feature type (S100_FC_Item:code), primitive type (S100_FC_SpatialPrimtiveType), and unique feature ID. The Lua portrayal scripts will call into the application to retrieve attributes, spatial elements, and associations as needed. More generically, the application will provide a dataset or group of datasets to the portrayal, and the portrayal will request features, spatial elements, etc. as needed and as determined by any Interoperability Processing which may be applied. This process is shown in figure 4.

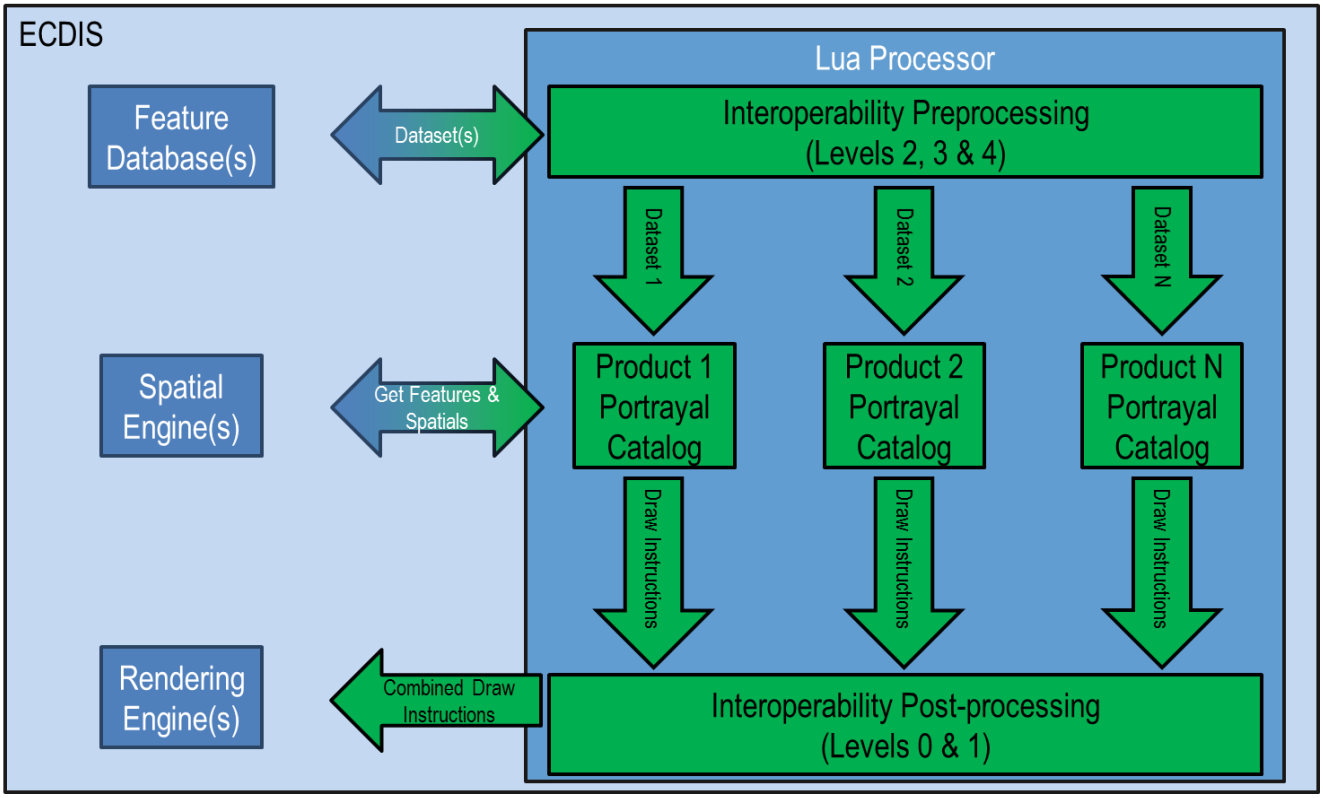

Figure 4 - Lua portrayal process

Similar to the Lua portrayal, the proposed Interoperability Processing design will be able to call into the application to request the information it needs when it needs it. This will greatly simplify implementation. The Alerts and Indications model will also benefit from this capability. Table 1 may be expanded slightly to support Alerts and Indications as well as Product Interoperability requirements.

Note that the two changes described in this paper are backwards compatible with the current XSLT portrayal. A "translator" application can be written which mimics the behavior of the current XSLT processor. This translator would accept a portrayal input schema and output the current portrayal output schema. The translator would utilize the Lua portrayal to generate the output schema, and the translator would implement the functions required from table 1.

Using a translator in this manner would allow manufacturers supporting legacy applications to easily interface with the Lua portrayal. Additionally, they would gain the ability to modify the input schema and translator application to more closely match their desired SENC format as the portrayal input schema would no longer be dictated by the product specification. An application using a translator in this manner instead of interacting directly with the Lua portrayal would likely not be able to use the Product Interoperability or Alerts and Indications.

In order to demonstrate the efficacy of the described Lua portrayal, we are providing as a companion to this paper:

1. A representative Lua portrayal catalogue along with scripts for most S-101 CSPs, including DEPARE03. (Note: DATCVR02 will be implemented as part of multi-product rendering / product interoperability).
2. A proof-of-concept implementation of the Lua portrayal, implemented in the S-100 simple viewer application.

If desired, we will provide prior to the next working group:

1. Recommended editorial changes to S-100 part 9.
2. Source code samples for embedding a Lua interpreter, provided in C++, C#, and Java. Samples for additional languages can be provided on request.

3. Source code samples for the functions described in table 1.
4. A Java implementation including source code of the translator application described above.

All items will be made available for download from the basecamp S-100 Test Bed website at www.basecamp.com.

Recommendations

1. Augment or replace the use of XSLT within the S-100 portrayal with Lua as described.
2. Update S-100 Part 9, including removal of the portrayal input schema.

Action Required of S-100 Focus Group

The group are invited to:

a. Note the paper
b. Compare the portrayals generated by the two versions of the S-100 Simple Viewer application
c. Examine the provided source code samples
d. Evaluate the provided Lua S-101 portrayal catalogue
e. Discuss the recommendations