Paper for Consideration by DIPWG

New Direction for S-100 Portrayal:

Proposal to Adopt a Declarative Approach for ECDIS Portrayal using OGC Symbology Encoding (SE) and Filter Encoding (FE)

Submitted by:	Jeppesen
Executive Summary:	In the last few months a draft has been written for a portrayal part for the next version of S-100. This draft is now in a state where its impact can be better assessed. Having done an assessment of the Portrayal Part there is now increasing concern about the direction it has taken. The concern is that the current draft is not desirable for the IHO, e-Navigation or the industry at large. This paper therefore explains the concerns, proposes an alternative direction to which we are willing to commit to do the necessary proof-of-concept work. And if the proof of concept is accepted, we will prepare part 9 of S-100.
Related Documents:	None
Related Projects:	S-100 Portrayal (Part 9)

Background

S-100 is built on existing standards, ISO TC211 in particular. This was done to facilitate interoperability[1] with other GIS communities, since S-57 had demonstrated to be very hydrographic community specific. This very narrow focus resulted in a need to develop the majority of software tools needed for creation, display and use from scratch, and thus driving up cost and delaying adaptation. The path followed in the development of S-100 v1.0.0 has shown that this focus on facilitating interoperability has been recognised as S-100 was chosen by the International Maritime Organization (IMO) as the base architecture for all navigational related e-Navigation information[2].

Introduction

The development of the S-100 portrayal part was not included in S-100 v1.0.0 for many reasons, among them concerns that the ISO TC211 portrayal standards did not meet the needs of the International Hydrographic Organization (IHO). In the last few months there have been developments with a draft being written for a portrayal part for the next version of S-100. The draft is now in a state where the impacts can be better assessed. Having done an assessment of the Portrayal Part there is now increasing concern about the direction it has taken. This concern centers on how the chosen approach and the focus on ENC is likely to lead to similar problems with adoption delay as S-57/S-52 faced, and moreover incompatibility and inconsistencies among implementers along with higher than necessary costs.

[1] http://www.hydro-international.com/issues/articles/id1274-The_IMO_eNavigation_Concept_and_the_IHO_S_Data_Standard.html

[2] http://www.uscg.mil/imo/nav/docs/nav57-report.pdf

Problems with the proposed portrayal specification

Considering the S-100 spirit of compliance with other GIS standards, the proposed portrayal specification seems out of sync as it is mainly a whole new and untested creation with minimal reference to other standards.

The proposed draft portrayal part has mainly an *imperative*[3] approach to defining portrayal rules and includes a newly defined programming language. The disadvantage of using a newly-defined programming language is that it will require system vendors each to implement it, with the result being incompatibilities and inconsistencies that may not be realized or recognized until the S-100-based ECDIS systems are fielded. This would also be the case for any system using an S-100 based product (for example ice, inland ENC, nautical publications, IALA AIS-ASM, etc), which makes use of the proposed portrayal. Inconsistencies in programming languages arise from a number of factors:

- Implementations of the compilers and runtimes that must be ported between different versions of the host operating system.
- Implementations of the compilers and runtimes that must be ported between different host operating systems.
- Implementations developed by different teams or companies.

In the case of the proposed draft, all of the above factors will be in effect as different hydrographic offices use different versions of operation systems, OEMs use different operation systems to build their ECDIS and there are many manufacturers making systems for producing data and using data. This complexity will increase with the adaptation of e-Navigation. Furthermore, no modern programming language that we know of has ever been implemented multiple times without incompatibilities, even where the specification and implementations are made by experts. These are a few recent such examples:

- The C and C++ languages have been standardized for literally decades and still we see compatibility problems between compiler and runtime implementations, so much so that many features (for example, templates, exceptions) are considered non-viable for portable code running in mission-critical systems. These are standards that are defined by programming language experts and implemented by mature compiler development organizations.
- The Java programming language, again specified by experts, saw compatibility issues between the official Sun, IBM and BEA implementations due to the different garbage collection and reflection implementations.

- [3] Imperative programming specifies instructions that must be executed in sequence. It explicitly states how a result is obtained and often indicates what technologies or techniques it is dependent on.

- The Ruby language, starting with a robust reference implementation, has been re-implemented in Java (JRuby) and .NET (IronRuby). In both cases it took years for the JRuby and IronRuby teams to reach a level of compatibility suitable for the direct porting of production applications.

The above does not mean that ECDIS OEMs will require the same effort, but it illustrates the difficulties and complexities of creating multiple implementations of a programming language. Furthermore, this will not be made easier with further development and subsequent adaptation of e-Navigation. We should assume that a novel programming language specified by non-experts will be subject to numerous problems. These problems are manageable within an organization where only a single implementation is needed, but they are very difficult where multiple organizations are involved.

The current draft is envisioned to become an extension of S-100, but it focuses almost entirely on the ENC scope. It needs to be broader, particularly when considering e-Navigation developments. For example, consider the possibility of packaging the symbolization with the data, i.e. when symbolization is originated by the data provider instead of the technology provider; the imperative language creates a safety and performance threat for the end-user application. It is easy to imagine a situation where the symbolization instructions, provided with the data, are prepared in a suboptimal way so its execution slows down the application or even leads to an infinite loop (system freezes). Looking at the sample portrayal catalogues given with the draft document, one could say that they are programming language instructions parsed into XML form. For example the BCNCAR case: 4 lines of S-52 Look-Up Table (LUT) now become 500+ lines (273-797) of rather complicated code (which even contain loops). This leaves much room for errors and requires a great deal of code-optimization.

We have questions and great concerns about how mature the proposed imperative language is. We believe that it contains syntactic flaws (see Annex A) and that it lacks development tools such as debuggers and profilers. This lack of development tools will mean that any implementation of the current draft will be cumbersome and costly industry wide, thus hindering adaptation of S-101 and that it may also hinder the development of e-Navigation. It is our opinion that the disadvantages listed above are so significant that a new approach is needed.

A new approach

A *declarative* [4] approach would better allow for the optimization of graphics performance. A declarative definition provides simpler and clearer structure of the resulting graphical primitives, which makes it easier to discover spatial relationships; it also allows to process that structure in order to reduce the data needed to be displayed on the screen. The S-52 LUT is an example of a declarative approach. The declarative definition allows for better verification and analysis of its own structure; this is important in the case when the symbolization is created or tuned by the data provider or even by the end-user

- [4] Declarative programming specifies a set of rules about what outputs should result from what inputs. It does not state how the result is obtained and does not indicate what underlying technologies or techniques should be used.

instead of the software solution provider. The declarative symbolization also allows for the creation of visual tools for editing and updating in WYSIWYG mode. On the other hand, the *Imperative* definition must be prepared mostly by hand, thus being more error-prone. For more information on the declarative versus imperative approach, see Annex B.

A few different approaches have been attempted while the portrayal part of S-100 has been under development. Some will remember Geomod's work. However, we feel that not all possibilities were given a fair review before being dismissed. Importantly, modern standards for portrayal based on XML were rejected. We believe this was a mistake, and specifically propose basing the S-100 portrayal on the Open Geospatial Consortium's (OGC) Symbology Encoding (SE) and Filter Encoding (FE) specifications.

Some reasons for dismissing these standard solutions were:

1. Modern templating tools are tied to XML and GML.

   Rebuttal: The OGC SE standard assumes the feature and attributes (termed properties in the specification) are encoded in XML; this assumption is not necessary. What is important is that the S-100 feature model can be mapped into the OGC (GML) model. A GML encoding is intended for the S-100 standard; at this level the OGC SE should be compatible. The proof-of-concept will determine if any gaps exist.

2. The standards are too complicated for users.

   Rebuttal: The OGC SE standard is in current use by cartographers and other subject-matter experts in other industries. They are optimized for specifying the portrayal of simple features and are extensible to the portrayal of complex features. We believe this is a much better fit for S-100 than a novel and newly defined programming language.

3. The standards are too complicated for vendors and would burden them by requiring the implementation of full support.

   Rebuttal: The OGC SE standard has existing, and in some cases open-source, implementations. It is possible that we will wish to identify a profile of the standard in order to simplify its use in S-100 (as was done with the ISO TC211 standards in several cases). That will be a matter of study.

4. The standards do not support operator or system presentation settings such as needed for ECDIS.

   Rebuttal: OGC Filter Encoding, on which OGC SE is based, supports the extension of functions and operators. S100 would define the necessary extensions needed for presentation settings and these would be implemented as part of compliant systems.

5. The standards cannot express all of the Complex Symbology Procedures needed for ENCs.

   Rebuttal: This concern is correct but we believe is manageable as OGC SE and FE are extendable

and new functions and operators can be added. Note that only a very small number of ENC CSPs are not directly supported using OGC SE/FE. Fundamentally we believe that the presence of a small number of CSPs needed for a single (albeit the most important) product specification should not drive the entire approach for the portrayal specification for everything S-100 based, particularly when the alternative that has been offered has so many liabilities.

As such, treating the non-supported CSPs as special cases requiring special handling is appropriate. The current practice where the few unsupported CSPs (or their subfunctions) are documented as part of S-101 and implemented by the vendors would work, and discourage other product specifications from overly-complex portrayal rules that fall outside the mainstream of GIS systems.

We believe a solution to the S-100 Portrayal will have the following attributes, most of which are consistent with the rest of the S-100 specification and its goals:

1. Be based on open standards that have been embraced by the GIS user community and its vendors.
2. Be simple for the vast majority of portrayal problems that are simple; only introduce complexity where needed and not as the baseline for all portrayal.
3. Be declarative to avoid coupling the portrayal catalogs to implementations that prevent vendors from innovating and optimizing solutions.

OGC SE & FE meets these goals and should be considered by TSMAD as a basis for portrayal:

1. It is an OGC standard and is harmonized with other ISO and OGC standards. It has been implemented in several GIS systems.
2. Its markup for simple portrayal is straightforward, but it incorporates optional capabilities for complex rules based on attributes as well as temporal and spatial data.
3. It is fully declarative with extant independent implementations and extensions for the rare cases that can not be modeled through the specification.

Background: OGC Symbology Encoding

OGC SE is an XML language for defining the portrayal of features and coverage data. Though originally intended for use with web servers such as Web Map Servers (WMS), its generality was recognized and it was separated from the Styled Layer Descriptor (SLD) Profile of the Web Map Service. This latter specification now forms a companion document to OGC SE.

OGC SE defines styles which bind a feature or coverage type to rules. Rules declare how the type is to be symbolized. OGC SE uses the OGC Filter Encoding (FE) specification which defines an XML predicate language. That language covers not only attribute-based filters but temporal operators and a broad range of geospatial data and concepts. Using filtering encoding, rules can conditionally use different symbolization to portray features and coverage data.

OGC SE has been broadly adopted and implemented as part of SLD systems:

1. GeoServer: an open-source Web Mapping Server (1)
2. AtlasStyler: a standalone free application for editing SLD/SE (2)
3. Arc2Earth (3) (commercial) and Arcmap2sld (4) (open source) allow exporting ArcGIS style definitions to SLD
4. QuantumGIS has two plugins allowing to export the style definitions to SLD as well

The first application is the OGC reference implementation for their web standards. The second application is a two-way SLD editing tool, i.e. it can both read and save after editing the SLD files. The other tools provide a different workflow: they assume the styles are created in GIS applications with respective GUI features and then are exported to SLD. This demonstrates that SE can express the symbolization rules of those elaborate applications.

(1) http://geoserver.org/display/GEOS/Welcome

(2) http://en.geopublishing.org/AtlasStyler

(3) http://www.arc2earth.com

(4) http://wald.intevation.org/projects/arcmap2sld

An OGC SE-based S-100 Portrayal

Definition of an SE-based portrayal specification would follow analysis and demonstration of its use for marine cartography. Specifically, Jeppesen will undertake to demonstrate at TSMAD24/DIPWG4 that an SE-based portrayal specification can be used to portray ENCs using the S52 standard. The demonstrator would show that:

1. Example S52 look-up table entries easily translate to OGC SE.
2. Example S52 CSPs that include basic logic can be represented in OGC SE
3. Example S52 CSPs that include the most complex rules and computations can be handled through well-described extensions.

An underlying assumption is that CSPs may be supported by addition feature data that is readily computed in production. (This assumption is made for the programming language as well.)

If the new approach is accepted, Jeppesen will commit to draft the S-100 portrayal specification, which would, when finished, include the following elements:

- The S-100 data model would be mapped to the OGC feature model. Note that this doesn't require conversion of S-100 features into XML elements; vendors can choose to implement the mapping in a manner that suites the system needs and performance constraints.
- The extensions needed for the most complex CSPs.
- If it is determined there are portions of OGC SE that are not needed and impede its use or implementation within ECDIS system, a description of a profile that excludes those portions.

Annex A

On how the imperative approach will lead to incompatibility and inconsistencies:

- The choice to use a newly-defined programming language will require that system vendors each implement it, with the result being incompatibilities and inconsistencies that may not be realized or recognized until the S-100-based ECDIS systems are fielded.

- Inconsistencies in programming languages arise from a number of factors:
    - Implementations of the compilers and runtimes that must be ported between different versions of the host operating system.
    - Implementations of the compilers and runtimes that must be ported between different host operating systems.
    - Implementations developed by different teams or companies.

- In the case of the proposed draft, all of the above factors will be in effect.

- No modern computer language that we know of has been implemented multiple times without incompatibilities, even where the specification and implementations are the product of experts. Examples:
    - The C and C++ languages have been standardized for literally decades and still we see compatibility problems between compiler and runtime implementations, so much so that many features (for example, templates, exceptions) are considered non-viable for portable code running in mission-critical systems. These are standards that are defined by programming language experts and implemented by mature compiler development organizations.
    - The Java programming language, again specified by experts, saw compatibility issues between the official Sun, IBM and BEA implementations due to the different garbage collection and reflection implementations.
    - The Ruby language, starting with a robust reference implementation, has been reimplemented in Java (JRuby) and .NET (IronRuby). In both cases it took years for the JRuby and IronRuby teams to reach a level of compatibility suitable for the direct porting of production applications.
    - The list could continue with JavaScript/ECMAscript, CORBA, etc

- The above does not mean that ECDIS vendors will require the same effort, but it illustrates the difficulties and complexities of creating multiple language implementations.

- We should assume that a novel language specified by non-experts will be subject to numerous problems. These problems are routine and manageable within an organization where only a single implementation is needed. They are very difficult where multiple organizations are involved.

- The problem of inconsistencies between implementations can be partially addressed, but not without issues and costs. One solution is to specify a reference implementation. The reference implementation would have to be owned and controlled by the IHO with source-code licensing defined that does not create an anti-competitive landscape for industry. Alternately, a test suite could be devised and source-licensed. In either case there will be additional development and management necessary to support the standard.

On the implications of a non-standard, novel and imperative portrayal solution to the IHO and the industry:

- IHO will see a need for changes to the specification after its approval as specifies and implementers identify problems and omissions that need to be addressed in order to assure conformity and consistency between implementations.

- Organizations that create S-100-based product specifications (IHO, IEHG, etc) will need to develop the capability to code and test their specifications, rather than simply specifying the rules for portraying their data. This leads to questions regarding what implementation(s) will be used by these organizations and who will be responsible for verifying the portrayal catalogs will run (correctly) on different implementations.

- Implementers will have to invest in new technology that is only applicable to S-100 systems, rather than being able to reuse or investing in standards-based technology.

- Implementers will have limited ability to innovate in their solutions, and to optimize their solutions for the platforms chosen by integrators for their ECDIS systems. For example, the specification around the graphics primitives and their sequence of calling may not be suitable for the graphics systems of different platforms (OpenGL, DirectX, GDI, etc.) The binding of graphics should be determined by the vendors based on a specification of what the portrayal should look like, not a specification of the implementation.

- Classification societies will have to manually interpret how the algorithms are executed to determine whether the results in an ECDIS are correct. This will increase the effort for type approval while decreasing its reliability.

- Users will see less stable and less consistent display of data as a result of having multiple implementations of this new programming language. Systems may be less performant due to the restrictions imposed by the language.

On the proposed language itself:

- The recasting of the original scripting language into XML does not change its imperative nature, and in fact makes the language less readable and more bulky.

- Specific high-level issues on its definition. I've submitted numerous lower-level issues as well against the first and second drafts but have not verified whether the new draft resolves them (no indication it did).

    - The use of the context is under-described and problematic. 1. It appears that the context variables are in scope at all times, but it's unclear what hiding and shadowing rules apply. 2. It's unstated whether the context variables can be changed; we'll assume not but otherwise there are significant questions about safety in parallel implementations. 3. Allowance of global visibility is a strongly deprecated approach because of the problems it introduces in writing and understanding the code. 4. The context is treated as a global store but is modified locally. Is the modified context available to all executions of rules in the ruleset if parallel processing is being used? 5. Rules for how context parameters are presented and managed between multiple active product specifications need to be decided. 6. Not all context variables that will be desired for the rule writers are going to desire will be appropriate for user modification; appropriate visibility management is needed.

    - As proposed, the language includes S-100 modeling concepts as part of its grammar. The implication of this is that changes to the data model could force changes to the presentation specification at the lexical level. This may lead to situations where changes to the S-100 model must be assessed with respect to the impact of the presentation implementation technologies. Changes to the grammar are much more impacting than changes to foundation-classes. An answer to this risk would be to generalize the language to support user-defined types, but in that case we should favor using a modern, mature, general-purpose language designed for that.

    - The proposed language lacks any error handling facilities, a serious concern for a technology that will be deployed to operational environments. The policy appears to be to silently ignore errors in order to be robust; I believe that is misguided and will result in hard-to-diagnose problems, both in development and in operations. Notably, in concert with our stated concerns about the lack of a tool chain, the language does not even specify the equivalent of a 'printf' to allow primitive debugging during development.

    - The language defines subroutines but doesn't define any scoping or importing syntax and rules for linking. Presume the intent then is that all the rules form a single 'compilation unit'; if so should be stated and the ordering dependencies should be stated. Judging by previous experience with other languages, lack of first-class scoping and importing facilities will create problems in rule-code management.

- All of these deficiencies, along with concerns about implementation conformance, lead to a conclusion that if it was the intent of the standard to rely on a programmatic, imperative approach, the best alternative would be to use an existing, mature language and define a standard API. Python has been independently identified by several industry contributors as a good candidate as it is mature, portable, extensively documented, and well-supported with development tools. More importantly, it was designed with non-programmers (specifically, SMEs) as its intended user base. This matches exactly with the use case for S-100 portrayal specification.

Annex B

On imperative versus declarative

Imperative languages are algorithmic i.e. they transform a given input to the desired output by specifying a sequence of actions to be performed on the input i.e. they tell you "how to do" something. In contrast declarative languages simply specify the desired output for a given input. They do not spell out the exact sequence of steps which need to be carried out to achieve this. All they describe is the final outcome (i.e. they are similar to a requirements specification) and the "how to" knowledge is embedded within the engine/implementation.

This allows an implementer to concentrate all the optimization techniques in the engine. Many kinds of optimizations can be done. For example, concurrent or parallel programming makes use of modern multi-core processors. Writing concurrent code that runs different threads in parallel is not an easy task; while this can be done by the engine developer, it cannot be expected from symbolization authors. Another optimization technique is the execution of some portions of the code in the graphical processing unit (GPU) as modern graphic cards are much faster in certain types of calculations than the CPUs. But instructing the software to use the GPU computational power is an even more complicated task and again we cannot expect that the symbolization programming language is compliable with CPUs and GPUs as well.

There are examples of declarative languages from other domains that show advantages of the approach:

1. *SQL selects* are pure declarative and powerful language allowing the database servers to optimize the execution of the queries by analyzing the requests and transforming them into more effective structure.
2. Declarative languages for defining user interfaces from HTML/CSS to Microsoft XAML used in the latest .NET technologies allow rapid WISIWYG creation of the user interface (UI).

Wikipedia article on Declarative programming (http://en.wikipedia.org/wiki/Declarative_programming) adds additional interesting points. Please see a few quotations below and note the lack of *side effects* as a trait of declarative approach, the risk of infinite loop is an example of a side effect in imperative program.

> "Declarative programming is often defined as any style of programming that is not imperative."
>
> "A number of other common definitions exist that attempt to give the term a definition other than simply contrasting it with imperative programming. For example:
>
> - A program that describes *what* computation should be performed and not *how* to compute it
> - Any programming language that lacks side effects (or more specifically, is referentially transparent)"