# Summary of Discussion and Recommendations from the Production Framework Group: Dagstuhl Sprint, 16 – 20 October, 2017

## Overview

This document summarizes the issues, activities, and recommendations coming from within the Production Framework group during this sprint, and also reflects comments and questions which arose during presentation to and discussion with the broader group of participants in plenary sessions. Many important issues have surfaced during this work, and the impact of decisions made on the basis of recommendations here will impact the work of several groups and the DDI Alliance as a whole, potentially affecting the way in which not only the Moving Forward production work is conducted, but also the production work of existing production lines in future (DDI Codebook, DDI Lifecycle).

Because of this potential impact, it was felt that describing the issues and presenting them to those of the community not present at the sprint was particularly important, as some decisions go beyond the remit of the limited groups represented among the participants.

## Background

Colectica has produced an open-source platform for producing a model-driven standard, which supports the creation of XML schema, RDF, JSON, C# representation, and documentation. While this product – named COGS – is used by Colectica in their own work outside the DDI Alliance, they are long-standing members of the DDI community, and have licensed the tool in such a way that it is available to the DDI Alliance for use free of charge.

Dan Smith of Colectica presented the tool in an hour-long demo on Tuesday evening of the Sprint, to the plenary group. Members of the Production Framework team were already aware of the tool and its capabilities, as were many other members of the DDI community. This had led to questions around whether or not the COGS platform could be utilized to advantage by the DDI community in its own work.

As described below, some of the Production Framework team’s activity during the sprint was to conduct a risk analysis to determine whether COGS represented a viable alternative to the existing production platform. It is important to note that the COGS platform is heavily based on the capabilities of Github, a generic and widely used toolset for the management and storage of code during development projects. COGS takes this basic platform functionality and applies it specifically to the development and leverage of a model-driven standard, including the creation of multiple parallel bindings and documentation.

## Goals of the Risk Analysis Activity and Requirements

The risk analysis activity conducted by the Production Platform group had the following stated goals:

* Delivery of outputs (defined below) in a collaborative platform that conforms to the production requirements of a tool supporting development and management of a model-driven standard.
* The realization of a single platform for all production streams of DDI (DDI Codebook, DDI Lifecycle, etc.) to provide a uniform toolset for conducting common activities such as creation of schemas, documentation, etc. (as appropriate to that production stream).

In general terms, the risk analysis set out a number of requirements, and looked at relative costs and benefits of continuing to use the Drupal-based platform or moving to a COGS-based system. While Drupal was a good platform for early stages of the DDI Moving Forward development, allowing collaborative modelling and production of early versions of the documentation and bindings, it has some critical failings, especially in light of the approaching release of prototype and production versions of the standard. Many of these failings are related to the ability of the platform to support work processes such as validation and approval, version management, and “sandbox” functionality (enabling users to understand the impact of changes to the model before committing them to the production model).

A spreadsheet is available, produced by the risk analysis. The findings of that analysis are summarized here. Each requirement is presented as discussed, with a brief summary of pros and cons, along with additional actions which might be required.

### Versioning and Change Management

**Versioning:** In order to support versioning of the model in Drupal, additional development would need to be carried out on the existing platform. This would be likely to require a considerable effort, as the functionality we need is not native to the platform. The downside of a move to COGS is the lack of a visual editing interface for the models. This could, of course, be developed on top of the COGS platform (extent of effort to be determined).

**Roll back:** This is the ability to return to earlier versions of the standard when the current version contains mistakes or other unwanted changes. Drupal can roll back to the last backup, but not to a specific version (as it does not support versioning, which COGS does.) There is no particular cost to utilizing this feature of COGS – it is inherent in the management features of Github.

**Differencing Changes:** Another capability of COGS (again courtesy of tools integrated with Git) is the ability to compare two versions at a granular level, and to easily visualize what has changed. This functionality also includes the merging of a changed resource into the existing library when creating new versions. Drupal has no such functionality.

**Approval Process:** The platform must support the process of inspecting and approving/rejecting proposed changes, with approved changes being merged into the production line of the model. In Drupal, we have the ability to include or exclude entire packages (that is, sections of the library) from the production run which creates the documentation and bindings. This would need to be expanded to operate at the level of individual classes. With Git (as implemented in COGS), there is an “Accept /Pull Request” functionality that operates at the level of the individual classes in the library.

### Maintainability

**Skills:** The maintenance of the platform will of course be necessary. With the Drupal-based system, we require Drupal expertise, which has been available in the past, but is not incredibly common, especially as we have heavily customized the platform. COGS is written using .NET Core and C#. These are very common skill sets among engineers, as is the use of Git and integrated technologies.

**Access to the Platform/Portability of Infrastructure:** The current Drupal system and the associated production tools for transforms/build server are accessible, being hosted in the Cloud. Git is available as an existing hosted repository or as repository software run by users. COGS itself is cross-platform, as are the current technologies.

**Consistency and Conformance:** We have established rules for both the modeling/development of DDI, and for the various bindings and documentation outputs. Quality, consistency, and conformance checks (validation) can to a large extent be automated, and incorporated into the build process. This does not today exist on the Drupal platform. COGS offers a framework for writing validation rules using C#, and could probably incorporate other types of checks easily. The major advantage COGS has in this regard is that there could be real-time validation not just for entire production runs, but for both the model and all outputs at the time a change is committed to the repository, whether in the production branch or the “sandbox” branch. This minimizes the number of problems which will be introduced into the model as it is developed.

### Usability

**Interface for Content Modelling:** Business modelers and the Modeling Team both require an interface for working on the model, as do those involved in the QA and production process. For the former groups of users, a point-and-ckick GUI is most useful; for others, a batch interface (upload of input files) is more useful. Drupal has filled the need for a GUI up to this point. COGS does not have the visual editing features of Drupal, although they could be developed. Drupal lacks the batch interface which COGS provides (upload of CSV files describing the model). Both types of interface are desirable.

***[TOPIC FOR POSSIBLE DISCUSSION: COULD DRUPAL BE USED ONLY FOR VISUAL EDITING PURPOSES BY DEVELOPING A CSV OUTPUT? WOULD THIS MAKE SENSE?]***

**Collaborative Process:** We have a requirement for the platform to support a collaborative process. Drupal has had some issues with supporting this requirement historically (isolated changes cause problems). COGS will offers better tools to control this process, but lacks the collaboration features of Drupal in some ways in terms of the GUI features.

**Sandbox Capabilities:** Related to the collaborative process requirements are issues of separating the production model from the “in development” versions of that model. This requires a “sandbox” capability, which Drupal lacks. COGS offers the ability to have branches of the model, which can be merged into the production model when approved, and offers better support here.

### Notes on Requirements

Not all of the possible requirements are listed above, but the ones most critical to supporting the development of DDI work products at the current time (especially as Moving Forward shifts into a more production-oriented mode) are given.

It should be noted that – although not model-driven standards as such – DDI Codebook and DDI Lifecycle both have implicit models in the XML Schema which can be exposed if needed. Colectica has taken DDI 3.2 and transformed it into the CSV import files needed to load a model into COGS, and is able to produce several different bindings from that model as well as documentation. Thus, it can be seen that GOGS would – after the effort to migrate was made – provide a sufficient platform for all production lines of DDI, even though the focus of this discussion was on DDI Moving Forward. Drupal is specific to DDI Moving Forward, and presents no easy option for becoming a more general DDI production platform with support for DDI Codebook and DDI Lifecycle.

## Analysis and Recommendations

### Deliverables

DDI Moving Forward presents a shift in the overall deliverables: while DDI Codebook and DDI Lifecycle are delivered as XML Schema and documentation packages, the Moving Forward production line will also include some other deliverables. In consideration of what the Moving Forward package should contain, the following was identified:

* The Model – although this is at the heart of the documentation, we would also like to have a processable version of the model in XMI (an XML format for describing models). This will be in canonical XMI, a flavor of that format supported across many UML tools. There will only be an XML file for the entire model being published – there is no sub-set for each View, etc. as for some other deliverables.
* XML Schemas – DDI Moving Forward will provide XML descriptions of both the entire breadth of the standard (all objects in the library), and subset XML Schemas for each identified View (groupings of objects to support specific functions). These will be organized such that the XML Schemas are a coordinated set – an element created using a View schema would be valid according to the complete schema, etc.
* RDF-S/OWL Vocabulary for RDF – Our second binding to be delivered is one which would support the expression of DDI metadata as RDF. The delivered vocabulary would be monolithic, describing the entire model, and not have the supporting Views used for the XML. There will be a vocabulary for the entire library, and a set of View-level vocabularies. These would be expressed using OWL. Additionally, we may want to publish SHEX validations for each vocabulary, to allow schema-style validation of graphs created according to any of the published OWL vocabularies.
* Documentation - Documentation of the model is provided in two formats: as a clickable HTML presentation, and as a PDF document. Documentation would be organized such that each separate View would be documented in an abbreviated form of the overall documentation, which would cover the entire library. There will be both high-level documentation, explaining the whole of the specification and its supporting parts, and detail-level documentation covering each class, property, etc. **[ QUERY: The need for binding-specific documentation at the level of the entire library and each View is to be determined?]**
* Examples (including those specific to each binding) are important. These should be organized functionally (eg, by View) so as to make the model as approachable as possible for implementors. This presumes the existence of good examples which can be reused across the examples for different bindings.

***[TO BE DISCUSSED: CURRENT THINKING ON DETAILS OF BINDING-SPECIFIC DOCS NEEDS TO BE DETERMINED, ESP. FOR VIEW LEVEL. CHECK WITH JON.]***

* Mappings: As annexes to the main library specification documentation, we will provide object- and property-level mappings from DDI Moving Forward to DDI Codebook (v. 2.5) and DDI Lifecycle (v. 3.2). Also, we will include mappings against GSIM v. 1.1. While these mappings are currently in the documentation for each object/property (where applicable), this would provide a view of the entire library and how it maps as a whole.

### Recommendations

Giventhe goals, requirements, and risk analysis as described above, the working group at Dagstuhl came up with a set of recommendations for discussion within the broader DDI Alliance community.

It is felt that for long-term production, having a single development platform based on COGS would be very desirable. This shift away from the Drupal platform would need to be made carefully, and should not disrupt the development of currently scheduled releases (June 2018 Prototype Release).

The process management features (versioning, branching, support for approvals, etc.) provided by the COGS platform are a major deciding factor behind this recommendation. The lack of these features is becoming extremely problematic now that the development process in DDI Moving Forward shifts into a production-focused mode. The need for such features will not diminish over time, but will also exist in order to maintain the DDI Moving Forward specification over time (and will exist for other production lines as well).

After the June release, more attention can be paid to the issues raised by the risk analysis, and how they might best be mitigated.

Consideration should also be given as to how DDI Codebook and DDI Lifecycle might be migrated onto this development platform. While (as noted above) this has been proven to be possible, such a shift must also be coordinated with other ongoing work within the DDI Alliance.

The importance of examples as part of the supporting materials released alongside the documentation should be recognized. Good examples are a critical form of documentation for implementers. These might be focused on a single set of use cases applied across bindings.