Paper TT06

CDISC 360: Using Biomedical Concept Metadata to Generate Case Report Forms and Dataset Definitions

Sam Hume, CDISC, State College, PA, USA

ABSTRACT

This paper describes new metadata created by CDISC 360 and shows how to apply it toward the automated generation of basic study artifacts, such as ODM-based case report forms and a Define-XML specification. Specifically, it explains how an object-oriented Python program processes metadata from four sources to produce a Define-XML file. The four sources of metadata include: (1) Implementation Guide standards, (2) Controlled Terminology, (3) Biomedical Concepts, and (4) Templates. Biomedical Concepts and Templates represent new sources of metadata created by the CDISC 360 project. This paper uses visual representations of the metadata and logic to communicate the technical details. It also describes the role of the CDISC Library in providing developers access to this new metadata via its API. As an ongoing project, readers can influence these metadata constructs prior to final publication.

INTRODUCTION

The CDISC Foundational Standards define research data and metadata structures, but writing these standards as documents has yielded more text than metadata. Gaps in standards metadata limit automation opportunities. The inherent flexibility provided by the standards supports a broad range of implementations, but that flexibility also allows for inconsistencies that make scaling automation difficult. The lack of a conceptual foundation for the standards further contributes to these inconsistencies. The relationships that would be expressed by these concepts remain largely implicit in the current versions of the standards.

CDISC 360, a proof-of-concept project, implements a conceptual foundation to the standards metadata by providing the additional semantics needed to support metadata driven-automation across the clinical research data lifecycle [1]. CDISC 360 demonstrates the feasibility of standards-based, metadata-driven automation to help realize the primary benefits expected of the CDISC standards: substantially improved efficiency, consistency, and re-usability across the clinical research data lifecycle. These benefits drive the return on investment from CDISC standards implementations expected by CDISC stakeholders. CDISC 360 demonstrates end-to-end standards-based metadata-driven automation using three specific use cases [1]:

1. Produce a standards-based, machine-readable study specification.
2. Demonstrate the ability to generate study metadata artifacts, such as a case report form (CRF) or Define-XML file, given a specification.
3. Demonstrate the ability execute end-to-end data transformations to generate study data artifacts using machine-readable metadata.

This paper breaks down the metadata and technology that drove the CDISC 360 demonstration performed at the 2019 US CDISC Interchange conference. This demonstration showcased Use Case 2, described above, by generating a CRF and Define-XML specification file using CDISC 360 Biomedical Concepts (BCs) and standards metadata available in the CDISC Library. Since generating a CRF and a Define-XML file follows a similar process, this paper will focus on the *bc2define* application [4] used to generate the SDTMIG v3.2 [6] Define-XML [8] specification file for the CDISC 360 demonstration.

CDISC 360 METADATA

Four different metadata sources were used to generate the Define-XML specification file: (1) Implementation Guide (IG) standards metadata from the CDISC Library, such as SDTMIG v3.2; (2) CDISC Controlled Terminology (CT), such as the SDTM CT 2018-06-29 package [7]; (3) BC metadata; and (4) Template metadata. BC and Template metadata have been newly created by the CDISC 360 project, while the SDTMIG v3.2 and CT package represent metadata available via the CDISC Library API today [2]. The following sections describe the BC and Template metadata in more detail.

BIOMEDICAL CONCEPT METADATA

BCs address metadata gaps in the current CDISC standards. They provide the conceptual definitions supporting the existing CDISC Foundational Standards metadata. This conceptual metadata is necessary to generate operationally ready Data Elements (DE). These operational DEs represent the detail needed to create the dataset variable definitions and value level metadata needed to generate a Define-XML document.

A BC is a unit of knowledge created by a unique combination of characteristics. As noted above, BCs complement the existing standards, but omit the operationalization of the standards. That is, BCs exist independent of any given standards implementation, such as SDTMIG v3.2 or CDASHIG v2.0. A BC specifies an observation concept, or what should be observed for a specific subject assessment in a clinical study, but not how to capture the data or how to group observations together.

An observation concept consists of one or more Data Element Concepts (DEC) as defined in the ISO 11179 standard [3]. DECs represent the meaning of a variable and consist of a concept code identifier and a definition. DEs, or operational variables, consist of a unique pairing of a DEC and a Value Domain

(VD). A VD is the domain of possible values for a DE which include data types, formats, and constraints. A DE is formed when a DEC takes on a specific representation or VD.

For example, CDISC 360 creates a BC to represent the concept of systolic blood pressure. To effectively represent a systolic blood pressure measurement, we need data for the result itself, the units, the time of the measurement, possibly the body position, and possibly the laterality (Figure 1). We constrain the values for the measure to be a number and the units to those that represent this measurement, such as mmHg. Variables specific to systolic blood pressure do not exist within the CDISC standards. We apply the systolic blood pressure BC to the existing standards variables to create systolic blood pressure specific operational DEs that can be immediately deployed in a Define-XML file. We use Templates to provide the metadata that describes how to apply a BC to a specific version of an existing standard, as well as to group BCs together to support a specific context, such as a domain or dataset. Templates are further described in the **Template Metadata** section.

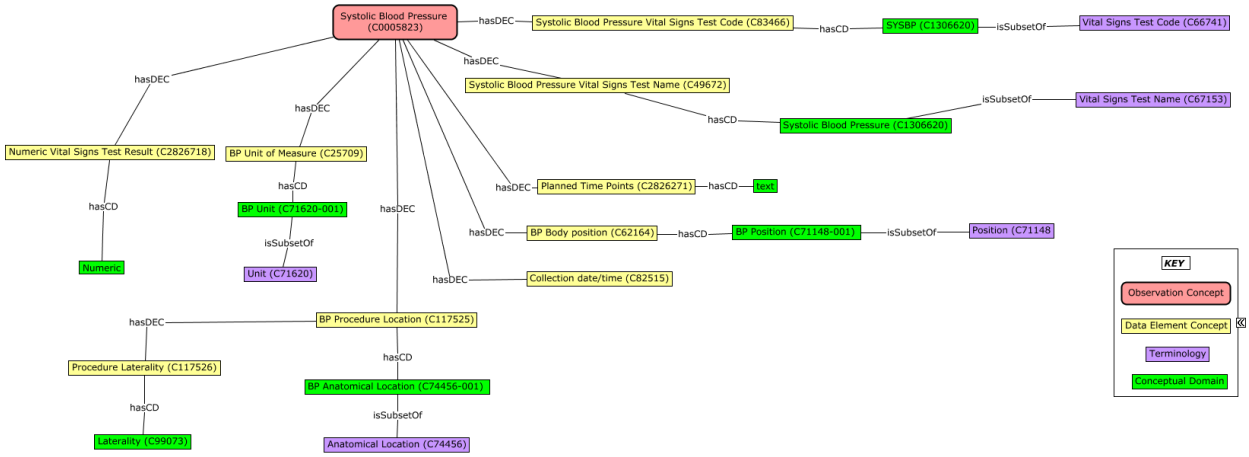

FIGURE 1. Concept map for the systolic blood pressure Biomedical Concept

A metamodel defines the structure of a BC. This model guides the implementation and processing of BCs by identifying the individual elements of a BC and how they are related to one another. The CDISC BC metamodel, illustrated in Figure 2, shows that a BC is defined by an observation concept, which is represented by a set of DECs that each have a Conceptual Domain (CD). The CD for a DEC represents the domain or the valid set of value meanings for a DEC. A CD can be enumerated with meanings or non-enumerated with a description.

A DEC is an abstraction of one or more DEs. As defined in the ISO 11179 standard and shown in Figure 3, a DEC is a specification of a concept independent of a specific representation. Thus, a DEC may be implemented by multiple DEs (Figure 4), which often vary in their VDs (Figure 5). For example, systolic blood pressure result DEs are implemented in the CDASHIG v2.0 and SDTMIG v3.2 using the VSORRES variable. Each DE addresses issues of

concrete representation, including codelists, units, and data types. The VD provides the concrete representation for a DE. A VD supplies the permissible values for a DE, and the CD provides the value meanings for the permissible values (Figures 3 and 5).

Two DEs that share the same DEC can be mapped to one another (Figure 4), such as when the DEs for CDASHIG v2.0 and SDTMIG v3.2 systolic blood pressure results both reference the same DEC. This is because DEs that reference a common DEC share the same meaning. Since BCs and their component DECs, as conceptual entities, are implementation independent, they are less likely to change over time than the standards that implement them.

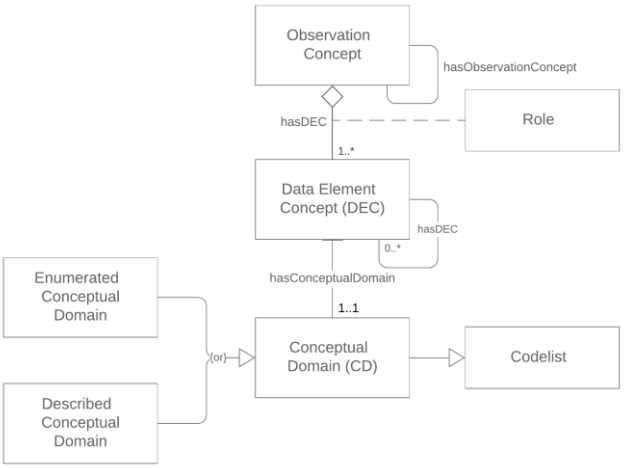

FIGURE 2. CDISC Biomedical Concept metamodel

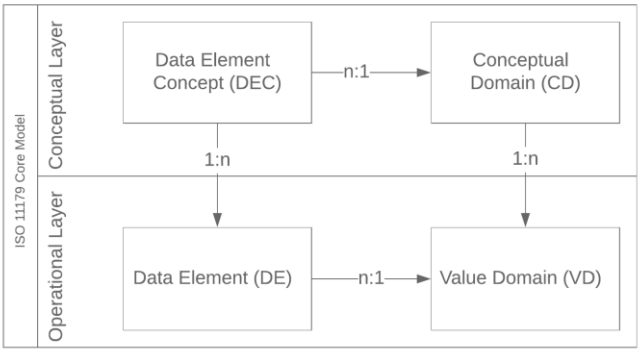

Figure 3. ISO 11179 core model

BCs, when combined with Templates, provide the metadata needed for software tools to automatically generate DEs, or operational variables, as part of a Define-XML file. They fill a metadata gap that currently each implementing organization must create on their own. The need for each organization to generate this metadata creates additional costs and delays for standards implementers, but it also creates variations in how the standards are implemented as each organization implements this metadata differently. Providing standardized metadata from CDISC Library in the form of BCs and

Templates will greatly reduce inefficiencies, reduce variability in standards implementations, and enable the development of new software tools.

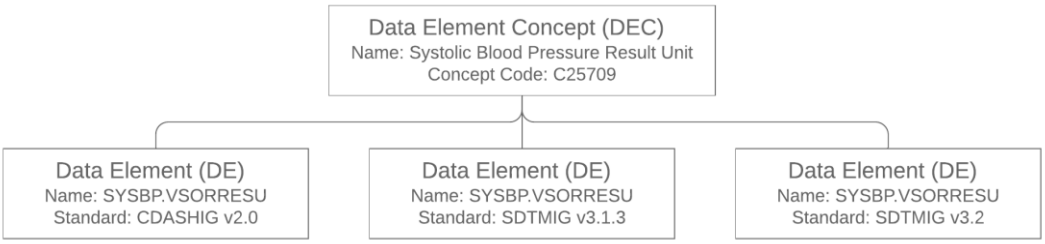

Figure 4. One DEC can be represented by multiple DEs

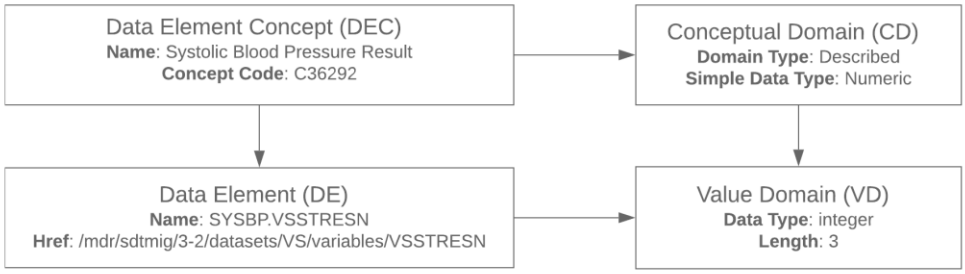

Figure 5. ISO 11179 core model example

CDISC CONTROLLED TERMINOLOGY CODELIST SUBSETS

The CDISC CT packages play an essential role in the application of the CDISC standards. For example, SDTMIG variable metadata in the CDISC Library contains a reference to the associated codelist for that variable if one exists. BCs function to specify the valid values for a variable in the context of that BC. For example, the SDTMIG v3.2 VSORRESU variable has the *Units for Vital Signs Results* codelist assigned to it, which is a subset of the main Unit codelist. However, for the BC *Height* the valid units of measure may only include two units, *cm* and *in*. CDISC 360 requires the instantiation of a codelist subset so that the BC can reference a codelist, which includes only the *Height* BC units. This subset will have an identifier such as a concept code. Formal subsets have not yet been implemented in the CDISC CT, but draft instances of codelist subsets will be created to support the CDISC 360 project. Final versions of CT subsets will be published in the CDISC Library sometime after the completion of CDISC 360.

TEMPLATE METADATA

The addition of BCs fill a gap in the current CDISC standards, but additional metadata is needed to create a Define-XML file or CRF for a specific context. We have the SDTMIG v3.2, SDTM CT package, and BCs, but we still need to know how to represent this content in a Define-XML file. Templates reference IGs, BCs, and CT and perform three major functions: (1) Showing how to bind BC metadata to specific IG variables, (2) Identifying the additional variables

needed to generate a dataset for a study, and (3) Specializing the VD for each DE, or operational variable, by constraining the definitions of the datatype, length, significant digits, codelists, and other metadata used to define DEs. Fundamentally, Templates apply BCs to an IG-defined dataset to dynamically create DEs for a specific study metadata artifact, such as a Define-XML v2.1 specification file.

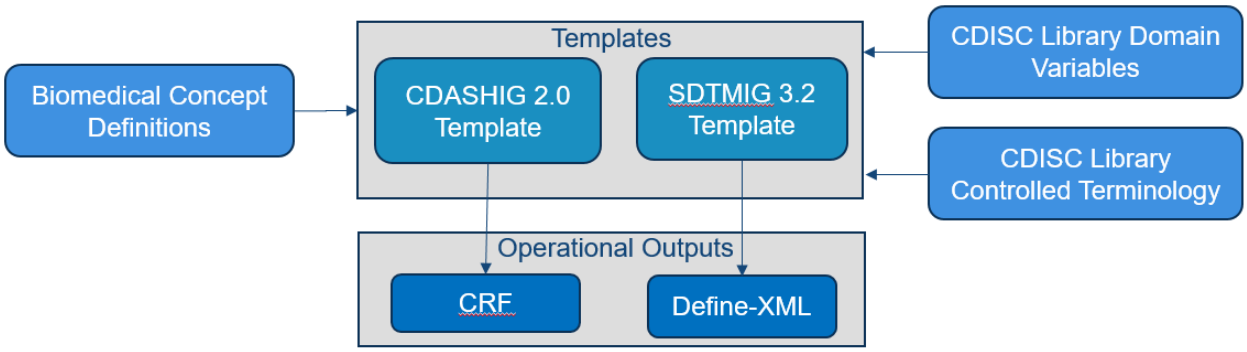

FIGURE 6. Using Templates to apply BCs to standards metadata

Figure 6 visually depicts how Templates provide the metadata needed to generate specific study metadata artifacts. Since Templates reference the other three metadata sources, they provide the metadata to drive the generation of the study metadata artifacts. They also provide metadata that targets a specific version of the metadata sources and the standards that specify the metadata artifacts, such as Define-XML v2.1. While BCs represent units of knowledge independent of any specific standard or version, Template metadata drives the creation of a specific version of a metadata artifact in the context of a specific study. For CDISC 360, that study has been defined based on the diabetes Therapeutic Area User Guide (TAUG).

BENEFITS OF A METAMODEL FOR BIOMEDICAL CONCEPTS

While metamodels introduce a level of abstraction that can be confusing to domain experts, the abstraction provides a number of benefits. The BC metamodel has three core classes with a total of seven classes (Figure 2). The total number of BCs created in production after the completion of the CDISC 360 may total in the tens of thousands. There will be a correspondingly large number of DECs and CDs. Using the metamodel, however, every BC can be created by a combination of those three fundamental classes. A software tool to facilitate the creation of BCs can be developed around this small number of classes, and does not need to know the semantics of tens of thousands of distinct concepts. Much of the validation of a given BC becomes a test against the metamodel.

The advantages of using a metamodel for BCs noted, there are also advantages to working with BCs in the context of the domain of clinical research. Domain experts typically prefer to work using their knowledge of clinical research and the CDISC standards instead of using the much more abstract metamodel.

These two approaches are not mutually exclusive. A domain-centric implementation in a graph database represents the metadata and relationships using language more familiar to domain experts. Queries can be written based largely on their knowledge of clinical research and the CDISC standards. In this case, the metamodel may still be part of the graph model even if the developer choses to ignore it. That is, the metadata contained in the graph database is composed of content created using the metamodel. Software developers benefit from this approach as they can work with the metadata using the metamodel or using domain semantics depending on the use case.

The CDISC Library makes use of this practice. When developers use the API to retrieve the SDTMIG v3.2, they receive content familiar to SDTM implementers, such as datasets, variables, codelists, roles, and other content encoded in the language of the standard. However, the CDISC Library model has been developed using an ISO 11179-based metamodel which simplifies the implementation of the CDISC Library platform.

AUTOMATION USING BIOMEDICAL CONCEPT METADATA

CDISC 360 demonstrated the generation of CRFs and an SDTMIG-based Define-XML file to show how BCs and Templates support the creation of study metadata artifacts (Use Case 2). The *bc2define* application implemented an object-oriented design in Python to generate the Define-XML file. This is just one approach to implementing the demonstration, and CDISC 360 will include others as the project proceeds. These tools will be limited and incomplete by design, including the *bc2define* application. Any software created for the CDISC 360 project is developed to test the application of the new metadata to drive automated end-to-end processing.

The process of generating a Define-XML file brings together all four sources of metadata: (1) SDTMIG v3.2, (2) the 2018-06-29 SDTM CT package, (3) BCs, and (4) Templates. Pulling metadata from this combination of sources represents a new process for most standards implementers. Participants in the CDISC 360 project have noted the conceptual challenge of bringing these metadata sources together to create a Define-XML v2.1 output. The conceptual challenge is due in part to the fact that creating DEs for use in Define-XML requires a multiple step process. To start, metadata from each distinct source must be retrieved. The Template metadata provides references to the three remaining sources of metadata, so Templates can be used to drive the retrieval of all metadata. The process of creating objects from the metadata sources is described in more detail in the **Factory Design Pattern** section. Figure 7 shows how the Template metadata references the BC metadata and applies the BC content to specific SDTMIG variables also referenced by the Template metadata. For example, the Template binds the diastolic blood pressure units from the BC to the VSORRESU variable. These newly created metadata objects are used to assemble the DEs, or *ItemDef* objects, used in the Define-XML file.

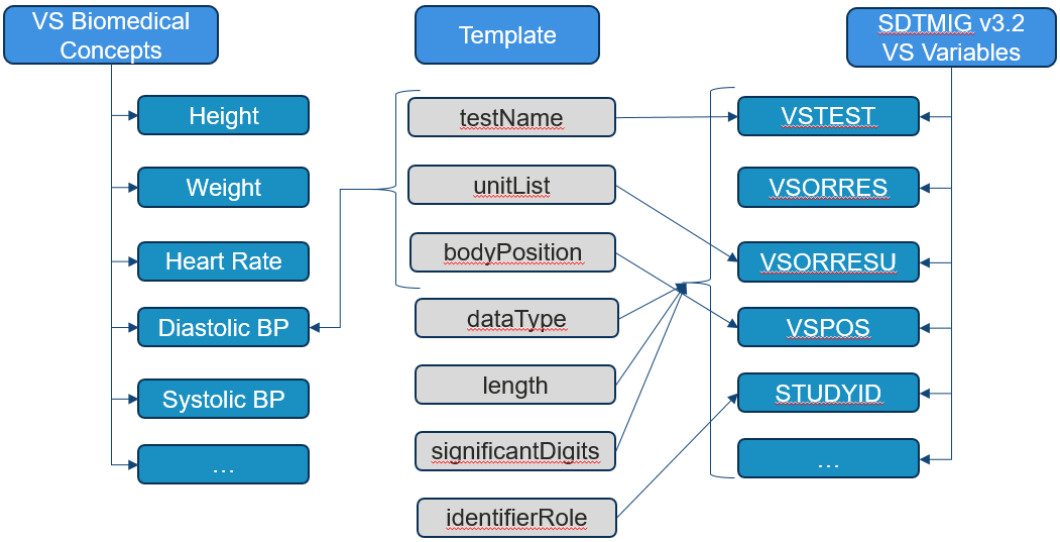

FIGURE 7. Applying Biomedical Concepts

The creation of the *ItemDef* objects follows a multi-step process, including creating the base item, adding a codelist, and applying each of the specializations included in the Template metadata. Figure 7 highlights how specializations, such as datatype and length are applied to specific SDTMIG variable definitions. The process of creating *ItemDef* objects is described in more detail in the **Builder Design Pattern** section. Once the *ItemDef*s have been created, they must be serialized as XML in the Define-XML document as described in the **Operational Data Model and Define-XML** section.

Once the CDISC 360 project is completed a programmer will be able to retrieve the metadata from all four sources from the sandbox CDISC Library. Retrieving metadata using the CDISC Library REST API is a relatively simple task. In CDISC 360 today, the SDTMIG and CT metadata are retrieved using the production CDISC Library API. More information on the CDISC Library API is listed in the **CDISC Library API** section. For simplicity's sake, the example Define-XML generated in this paper is constrained to the vital signs dataset.

FACTORY DESIGN PATTERN

A software design pattern is a general, reusable design that may be considered a best practice to solve a common problem. The factory pattern is one of the most widely used creational design patterns, and it encapsulates object creation by deferring object instantiation to subclasses. Encapsulating object creation is useful for creating BC and Template objects since the means of creating these objects will evolve over time. Today, we create BCs using files in the file system. In the near future, we will create them using the sandbox CDISC Library API. When BCs are finalized, they will be created from the production CDISC Library. As the means of creating these objects, as well as the structure of the objects themselves will evolve over the course of the proof-of-concept project, encapsulating object creation isolates the changes to a subclass with no changes needed to the main *bc2define* application. Without

the factory encapsulating object creation, if/else if/else conditional structures are needed to account for the variety of ways the BC and Template objects will be created in the *bc2define* application. The factory pattern approach simplifies the application code improving reusability and maintenance. Figure 8 shows the *bc2define* application making use of factory objects to create object representing the sources of metadata needed to build the Define-XML DEs.

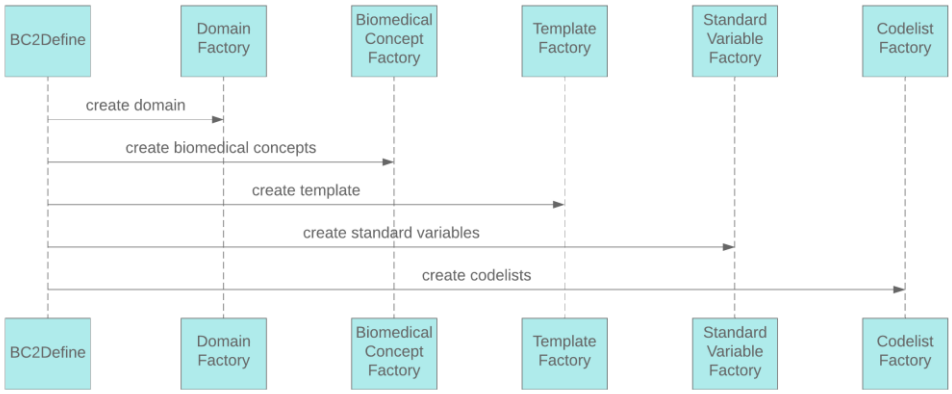

FIGURE 8. Sequence diagram highlighting the use of the factory objects

BUILDER DESIGN PATTERN

The *builder* design pattern is another creational pattern used to create objects. It works to simplify the creation of complex objects by implementing a divide and conquer strategy that implements the object step by step. It also makes the steps of object construction abstract so that different implementations of these steps can construct a variety of object configurations. Generating a DE, or operational variable object for Define-XML fits the description of a complex object as it takes a number of steps to combine the four sources of metadata mentioned previously.

We use the builder pattern to generate the XML elements needed to represent DEs for inclusion in the Defile-XML file. In Define-XML, variables are represented by the *ItemDef* element and referenced using the *ItemRef* element. The *ItemDef* elements are generated using the builder pattern. *ItemDefs* are created for dataset variables and value level metadata (VLM) variables, requiring two separate concrete builder classes. A simple class diagram for the item builder is shown in Figure 9. There are four basic class types in the builder pattern: (1) director, (2) abstract builder, (3) concrete builder, and (4) the product being built, in this case an *ItemDef*.

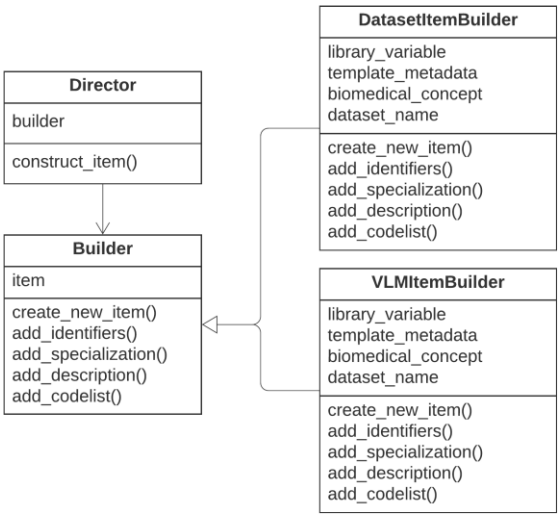

FIGURE 9. Simple builder pattern class diagram to generate *ItemDef* objects

Figure 9 shows two concrete builder classes that subclass *Builder*: the *DatasetItemBuilder* and the *VLMItemBuilder*. These two concrete builders reflect the *ItemDefs* defined for the dataset definition and for the VLM definitions in Define-XML, respectively. The director is instantiated using one of the concrete builder classes. Each of the methods defined in the abstract builder are called in sequence by the director to create the *ItemDef* instance that will be inserted into the Define-XML file, as shown in the sequence diagram in Figure 10.

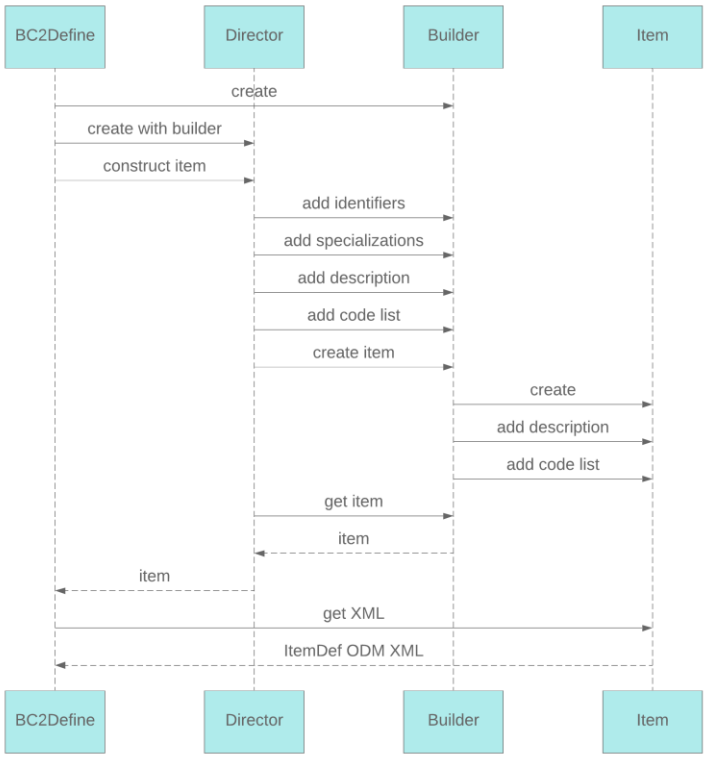

FIGURE 10. Builder pattern sequence diagram for VLM *ItemDef* builder

The director calls the methods in sequence to create the *ItemDef*s for use in VLM. The *add_identifier*s method is called first to create *ItemDef* identifiers, such as the *OID*, *Name*, and *SASFieldName*. Identifiers are required to instantiate the *ItemDef*. Next the *add_specialization* method is called to specialize the definition of the *ItemDef*, typically though the application of VD constraints. VD specializations constrain the values found in the base SDTMIG by, for example, providing a codelist subset, data type, length, or number of significant digits. The specializations are defined in the Template and in many cases the Template references BC content. For example, the valid set of units for a measurement result are typically represented by a relatively small unit codelist subset referenced in a BC. *ItemDef*s in VLM can be considered specialized versions of the dataset *ItemDef*s.

CDISC LIBRARY API

The CDISC Library provides a cloud-based CDISC standards metadata repository for the curation, management, and publication of the standards metadata in machine-readable formats, such as JSON, XML, CSV, and Excel. The CDISC Library provides a REST API for software applications to consume normative CDISC standards metadata in support of metadata-driven automation. Currently, all CDISC models, IGs, and CT standards metadata are available for retrieval from the CDISC Library API, simplifying access to standards metadata by software applications. The details on how to use the CDISC Library API can be found in the API documentation and Knowledge Base Articles (see **References**). As the new metadata specified by CDISC 360 are created as normative standards, this metadata will also become available via the CDISC Library API. During the course of the CDISC 360 project, new metadata, such as BCs and Templates, will be published as files or via a CDISC Library sandbox API.

OPERATIONAL DATA MODEL AND DEFINE-XML

As the final output of *bc2define* is a Define-XML file, the program must serialize *ItemDef*s and other objects into XML conformant with the Define-XML v2.1 standard. The Define-XML standard is itself an extension of the ODM v1.3.2 standard. Classes representing the various ODM and Define-XML elements, such as *ItemDef*, were implemented to support the creation and serialization of the Define-XML content.

PUBLISHING CDISC THERAPEUTIC AREA USER GUIDES AS TEMPLATES

In addition to supporting end-to-end automation, one unplanned, but possible outcome of the CDISC 360 project, could be a machine-readable way to publish the CDISC TAUGs. TAUGs represent an implementation of an existing standard that may influence future versions of the standard by proposing new domains or variables. Once the Foundational Standards, CT, and BC metadata have been developed in support of a TAUG, Templates could be published that represent specific CRFs and Define-XML dataset definitions. These would be

used to generate ODM and Define-XML examples to be published as part of the TAUG.

An immediate benefit of this approach would be machine-readable artifacts that enable users to directly implement a TAUG. In addition to publishing the BCs and Templates in the CDISC Library, CDISC could use programs like *bc2define* to generate ODM and Define-XML TAUG artifacts that could be used by implementers. Once vendors implement software tools that work with BCs and Templates, standards implementers could take a TAUG template and further refine it to represent an organization's therapeutic area (TA) standard. The TA Templates created by an organization could be further refined to represent the metadata for each study within that TA. Thus, the Template mechanism could be used to represent study level metadata, and this study level metadata could be developed by constraining and augmenting TA standards implemented using the same mechanisms.

ASSUMPTIONS AND LIMITATIONS

The CDISC 360 metadata and various approaches to automation are currently under development and will change over the course of the project, maybe significantly. Some metadata needed to support BCs do not yet exist. For example, concept codes do not exist for BCs and most codelist subsets do not exist. Furthermore, no normative standards content will be published by CDISC 360 so none of the metadata is available in the CDISC Library. Access to BCs, Templates, codelist subsets, and other novel metadata will be published in a CDISC Library sandbox to support the project.

Certain conventions are required to support the CDISC 360 metadata and its subsequent transformation into study-level metadata artifacts. For example, *OID* generation and naming conventions have been assigned to enable to creation of Define-XML elements such as *ItemDef*s. Conventions regarding which variables have associated VLM have been created.

To simplify understanding and ease maintenance as the CDISC 360 metadata evolves, the *bc2define* application generates a Define-XML file with one Findings domain. Support for datasets from different observation classes and for multiple datasets is currently being added.

The CDISC 360 project acknowledges, but has not yet addressed, the challenge of creating and maintaining a significant number of BCs and Templates. These additional metadata components may number in the tens of thousands when completed. After the CDISC 360 project, CDISC will need to train subject matter experts to create and curate BCs for publication in the CDISC Library.

CONCLUSION

CDISC 360 seeks to design and test new metadata to drive standards-based automation. The BCs and Templates fill metadata gaps in the standards and

are under active development in the CDISC 360 project. CDISC 360 seeks to create automated tests of this new metadata using a number of different technologies. This paper highlights how the object-oriented *bc2define* application uses the metadata sources to generate study metadata artifacts. Bringing together the four sources of metadata identified in this paper presents a more complex standards automation challenge than simply working with the existing CDISC Foundational Standards. It also enables a level of standards-based automation not achievable today. A version of this software, along with test BC and Template metadata, will be published in a CDISC Bitbucket repository by June 2020.

Regarding future CDISC 360 developments, the project is currently defining metadata to represent data transformations and derivations to support the flow of data through a study as well as deriving data values, respectively. The project is also creating Analysis Concepts (ACs) to complement the BCs and support automation of the analysis end of the lifecycle.

REFERENCES

[1] CDISC. (2020). CDISC 360 Project. Retrieved from https://www.cdisc.org/cdisc-360

[2] CDISC. (2020). CDISC Library. Retrieved from https://www.cdisc.org/cdisc-library

[3] ISO/IEC. (2013). ISO/IEC 11179 Part 3: Registry metamodel and basic attributes. Retrieved from http://metadata-standards.org/

[4] Hume, S. (2020). CDISC 360 bc2define. https://bitbucket.cdisc.org/projects/CDIS/repos/bc2define/browse

[5] CDISC. (2019). CDISC Library Service Desk Knowledge Base. Retrieved from https://wiki.cdisc.org/display/LIBSUPRT/CDISC+Library+Service+Desk+Knowledge+Base

[6] CDISC. (2013). Study Data Tabulation Model Implementation Guide v3.2. Retrieved from https://www.cdisc.org/standards/foundational/sdtmig

[7] CDISC. (2018). CDISC Controlled Terminology. Retrieved from https://www.cdisc.org/standards/terminology

[8] CDISC. (2020). Define-XML v2.1. Retrieved from https://www.cdisc.org/standards/data-exchange/define-xml

ACKNOWLEDGMENTS

Thanks to Ann White, Mike Hamidi, Bess LeRoy, Jon Neville, and Sally Cassells of CDISC for their thoughtful comments on this paper.

CONTACT INFORMATION

Comments and questions are valued and encouraged. Contact the author at:

Sam Hume shume@cdisc.org
CDISC https://www.cdisc.org/