Using the DDI 4 Process Model to Describe Historical and Prescriptive Processes

*Draft 0.1*

*Arofan Gregory, Jay Greenfield*

# Overview

This document is being produced as the result of some discussions at the DDI Moving Forward sprint at Schloss Dagstuhl in 2014. It is based on some earlier modeling work done by Jay Greenfield as part of the development of the core process model now found in DDI.

This core model was successfully used by the Data Capture group at Dagstuhl to describe data collection flows. This document makes a different use of the process model, attempting to describe historical business processes (Provenance) and prescriptive business processes – that is, describing the steps to be taken when a business process is being performed.

The intention is to produce two Views: one for generically describing an historical business process, and the other a View for generically describing a prescribed business process.

# The Current DDI 4 Core Process Model

The diagrams below give a simplified view of the existing core process model. In the first diagram, we see how process steps are extended to provide different kinds of process flow constructs.



In the second diagram, we see how Agents, Services, Inputs and Outputs are related to Process Steps.



This model is abstract, and is intended to be used by making extensions from the various objects it provides. For the Data Capture group, the Process Step object was extended to describe the flow of an instrument for data capture.

# Business Process Scenario

We can use a single business process to test the use of this core model for both our historical and prescriptive cases. In the figure below, we have a common business process – developing a questionnaire.



The top three boxes in this diagram can be ignored – these reflect the core process model, and simply indicate that this particular process is a single, sequential process.

Here, we take a hypothesis, identify what constructs are needed to test that hypothesis, identify the measurements we need, and then assemble these into first a human-readable questionnaire (presumably a Word document or similar), and then a machine-actionable questionnaire (a Blaise CAI file, for example).

This business process can be described as an historical process (one which has taken place) or as a prescriptive process (a description of how the process will be conducted). Each application of the core process model will be described in its own section.

This scenario is in fact quite simplified. As we address how the DDI model can be used to describe it, we will consider the potential additional complexities (for example, re-use of earlier questionnaires as a source of input; iterative steps within the process, etc.)

# Historical Business Process Description (Provenance)

For describing business processes which have occurred, for reasons of provenance and documentation, it is necessary to answer several questions: *what* was produced/altered? *Who* performed the alteration/creation? *How* was it done? *When* was it done? *What* specific inputs to the process were used? *Why* was the process performed? We will look first at a possible application of the core process model, and then explore how well this fits our example use case of questionnaire design.

## Applying the Core Process Model

Most of the objects and properties needed to answer these questions exist in the core process model. In order to leverage these, we will want to make an extension of the ProcessStep object.



As can be seen in the diagram, the ProcessStep object is extended by HistoricalProcessStep. This extension is critical, as it places a semantic on everythin g in the core process model: this is a description of a process *which has taken place.* This gives us the ability to describe what was done (the ProcessStep itself) , how it was done (the Service), and who did it (the Agent). We might also want to add a relationship to a ProcessStepDesign (see below) or to a step in an existing process model such as GSBPM or GLBPM.

We could use the generic Input and Output objects, but these will need to be specialized for historical use – they describe specific instance of the inputs and outputs, not generic types (see below). Further, we need information on when the action occurred, which can easily be captured by putting start- and end-times on the HistoricalProcessStep object.

One question arising from this modeling is whether Service should really be a specialization of ProcessStep, or an object in its own right.

Let’s look at Input s and Outputs. The requirement is to describe specific instances of whatever inputs and outputs were used historically. This can easily be done by refining the existing relationships between Input, Output, and ProcessStep.



Here, we have created two new objects - InputInstance and OutputInstance – which specialize the hasInput and hasOutput relationships inherited from ProcessStep by HistoricalProcessStep.

In practical terms, we have several potential types of input and output instances. Some might be metadata objects described in DDI, or groups of such objects. Others might be citable resources which do not exist in DDI form, such as external documents describing hypotheses, methodologies, etc.

This is easy to model using existing DDI objects:



Here, we see the Citation being available to both InputInstance and OutputInstance, as well as a relationship to a DDIResource object in both cases. The DDIResource object is a collection of one or more DDI-described objects. This gives us the expressive capability we need to specify the instances of inputs and outputs used historically in our process. It might be useful to add a type property to InputInstance and OutputInstance – probably a field which could utilize a controlled vocabulary.

## Our Example Use Case

In our example use case, we have one over-arching HistoricalProcessStep – questionnaire design. This has a sequence of related HistoricalProcessSteps: Review Literature, Research Date, Design, and Program. Each of these steps has a specified set of inputs and outputs, each a specific instance of a resource.

In ReviuewLiterature, we have shown in our simple diagram one input – a Hypothesis. In reality, there would also be a large number of inputs – each of the articles constituting the literature. All of these objects could be cited using the “externalResource” relationship between InputInstance and Citation.

The Service “Review Literature” is performed by one or more individuals (potentially by a set of intelligent software routines, but typically not under current circumstances), which are the Agents associated with the service. Thus, we can describe our *what,* our *how,* and our *who. When* is supplied by the start and end times on our HistoricalProcessStep object. *Why* could be a reference to external documentatiuon (an InputInstance) or be a description property on the HistoricalProcessStep object.

The output of our first process step are Constructs – these are the conceptual pieces which will be operationalized in our survey, and will most likely take the form of documentation of our analysis -0 these again could be identified using the Citation object related to OutputInstance, and are the inputs for the next process step.

Two potential issues emerge almost immediately: our Service objects is itself a subclass of ProcessStep, but not of HistoricalProcessStep, and our HistoricalProcessStep – design questionnaire – is a subclass of ProcessStep, but not of Sequence, which is what it actually is. A third potential problem is also evident – we cannot strongly type our input and output instances.

Are these real problems?

The Service object is a ProcessStep subclass, as the model currently stands. We may or may not have the right set of information here – what I want to know about a service is what it performs in business terms, not a description of the executed steps and sub-steps. We may need to create a Service object which is *not* itself a subclass of ProcessStep.

Our other potential issues are not actual problems: because a HistoricalProcessStep is a processStep, it inherits relationships with any other types of ProcessSteps, including sequences, which – because they are themselves ProcessSteps – can be a sequence of HistoricalProcessSteps, as is the case here.

If we have a type property on out InputInstance and OutputInstance objects, we can express the types of each input and output as needed. This is not strong typing, but it does express what we ideally want.

When we consider our next HistoricalProcessStep, we have a very similar picture: we take the Constructs coming out of the first HistoricalProcessStep (Review Literature) and we identify a set of Measures as outputs. In a typical case, these measures would be questions. They might be taken from the preceding cycle of data collection (an Instrument in DDI terms) or from a pool of questions. These resources could be described as InputInstances (if described in DDI) or as Citations (if not). The Measures which are the output of this HistoricalProcessStep are likely to be a set of DDI-described resources: variables, questions, and their response domains/representations. These can be identified using the DDIResource objects. The Service and Agent objects function as before, as do the start- and end-time properties of the HistoricalBusinessProcess.

The following two HistoricalProcessSteps – Design and Program – follow an almost-identical pattern, with the exception that it may be highly desirable to also describe their typically iterative sub-processes: a questionnaire is typically tested and revised several times, as might be the programmed CAI instrument. We might want to describe these HistoricalProcessSteps in detail, using the process model, and thus drill down an additional level. This would involve an approval process, using conditional-logic objects such as IfThenElse, Loop, etc.

This raises a question about how expressive the conditional logic description of our process steps is: do they describe business criteria, or only programmatic criteria? For this example use case, we need the former capability, at least in a documentary fashion. The conditional looping is not important here, assuming our process is linear in nature – what is important is referencing a prescriptive mocel, and then capturing which values were applied to the conditional logic to produce an historical record of hoe the conditional flow was traversed. We will need a link from HistoricalProcessStep to a ProcessStepDesign, and a way to capture the values used in the condtional logic specified in the ProcessStepDesign.

We can see that, using a set of extensions to apply the core process model, we can effectively describe historical business processes and provenance.

# Prescriptive Process Description

## Applying the Core Process Model

We will use a very similar approach when describing prescriptive business processes – first extending the existing core process model, and then adding whatever additional information is needed at theee level of the extension.

The prescriptive description of a business process is really the specification of the process itself – what in GSIM terms is the ProcessStepDesign. We will use this terminology as being more intuitive than the word “prescriptive”.

Out initial extension is again from the ProcessStep object in the core process model.



The extension of ProcessStep to ProcessStepDesign adds our primary semantic: what we are using the core process model to describe is the way a process is expected to be performed, as opposed to how an historical process was performed. This again makes available to us all of the existing relationships and properties of the core process model, such as Agent, Service, and all of the control constructs.

Where this model becomes very different from the HistoricalProcessStep model described above is in the Inputs, Outputs, and in the specification of Services. In a design scenario, the specific instances of these are not given – only restrictions about the tyope of the inputs, outputs, and (potentially) services. There is, however, an additional requirement that a specific instance of a resource might be required, if that is dictated by the business process, regulation, or system capabilities (Services).

Thus, we will need all of the expressive capabilities of our HistoricalProcessStep model, but in additioin, we will need to alternately specify types of resources. We thus will have a model which looks like:



The OutputType and InputType objects would carry fields to identify an external controlled vocabulary, the basis of which could be the set of identifiable DDI objects, but which would be supplemented by the types of real-world external objects which might also serve as inputs and outputs.

We have a similar situation with Services. Because we might want to specify a Service which implements a specific methodology or function, but not an instance of that service, we would need to add ServiceType to the mix.



We again have an object which could take a value from a controlled vocabulary, enumerating the types of services (based on HLG CSPA, GSBPM, GLBPM, etc.) – ServiceType. And ServiceTypes can have inputs and outputs, specified by instance or by type.

This again raises the issuer of how we describe our services, since they, too, can have inputs and outputs of either instance or type.

## Example Use Case

This follows the pattern established by the HistoricalProcessStep – we have all the expressive capability we need, but have avoided some properties (such as start- and end-time) and extended instead how we can describe types of inputs, outputs, and services.

Any sub-processes would here be described as conditional flows, rather than as linear paths through a flow. Otherwise, this example is again supported by the core process model.