

# Chapter 2

## Modeling Genome Data Processing Pipelines

Marie Schäffer

**Abstract** In order to conduct analyses on genome data, different calculation steps have to be done in a specific order, which constitutes a genome data processing pipeline. Still a lot of research is in process, in order to find faster and more reliable ways to do various analyses, so single steps or the whole sequence of the pipelines might be subject to change. A modular and flexible way to configure pipelines could simplify their use and the sharing of pipelines between researchers. With a possibility to configure pipelines without altering source code, bioinformaticians and technicians would be relieved of the task to rewrite a pipeline every time a single algorithm changes. This contribution proposes to use common process modeling tools for the abstract representation of genome data processing pipelines. The benefits and drawbacks of different process model notations are examined with special focus on the possibilities to specify execution semantics. As a prototype, a system for the parsing and execution of genome data processing pipelines specified in business process model and notation, is introduced.

### 2.1 Introduction

The analysis of genome data is an important element of both research and treatment of many diseases. Any advanced analyses consist of several steps that have to be executed in a specific order. The most fundamental of these analyses is the alignment, which assembles the genetic information from small DNA fragments. The succession of such analysis tasks can be represented as a pipeline and will be referenced as genome data processing pipeline henceforth.

Today, these pipelines often consist of a number of scripts calling each other directly. This approach has significant drawbacks: Every time the structure of a pipeline changes, a technician has to change the code of several jobs.

To achieve better maintainability, I propose a model-based approach for the definition of genome data processing pipelines. The objective is to deduce the suc-



cession of jobs from a graphical model of the pipeline. A scheduler that is responsible for the initiation and completion of jobs as described in Chapter 3 manages the execution. With such a system, every job would be stored once and get all pipeline specific parameters from the scheduler. Thus, algorithmic changes would only have to be incorporated in one script. Structural changes of pipelines could be incorporated without touching any source code.

Another goal of the approach described in this work is to enable external scientists and physicians to model their pipelines according to their individual needs and have them executed on a central computer cluster. I discuss the different requirements of the two user groups and introduce a corresponding research prototype.

The remainder of this work is structured as follows: In Section 2.2, different approaches for the modeling and execution of process pipelines are examined as well as other approaches concerning an easy interface for genome analysis processes.

Section 2.3 covers the steps needed to design executable pipeline models. First the general requirements for the representation of genome data processing pipelines are examined including the modeling of information flow and parameters for jobs. The different modeling notations are assessed for their applicability to the specified requirements. The degree of specification and standardization is taken into account as well as the question how intuitive user groups other than informaticians may use them. I describe one way to meet these requirements by using Business Process Model and Notation (BPMN) and applying an abstract view of the pipeline. The representation of such a BPMN model in the XML Process Definition Language (XPDL) is also discussed.

In Section 2.4, the research prototype is described in detail. I outline the desired functionality for programmers, researchers and physicians. The structure of the prototype is explained including the user interface, as this is an important factor influencing the benefit for the target group.

The evaluation in Section 2.5 focuses on the usability for the different target groups. Modeling decisions made for the prototype are reviewed and compared to alternative methods and the advantages and disadvantages of the approach used are compared to the systems discussed in Section 2.2.

In Section 2.6, I discuss the impact this approach could have and extensions that could be made to the basic system introduced here.

## 2.2 Related Work

A number of different notations for the creation of process models are used for different purposes. All of them have benefits as well as drawbacks for the desired usage. In this section, I introduce selected notations that might be fitting for the modeling of executable genome data processing pipelines.

Furthermore, I introduce approaches to the offering of a simple interface for genome analysis similar to my own.



## ***Petri Nets***

The concept of Petri nets is a system for the graphical modeling of complex process logic [65, Chap. 4.2]. It originates from Carl Adam Petri and focuses on parallel execution strands and their dependencies. A Petri net consists of places and transitions that are connected by directed edges. Graphically, places are represented as circles and transitions as rectangles.

Every place may contain a token. Depending on the specific type of Petri net, a place can also hold more than one token. The entirety of all places and their number of tokens represents the state of the Petri net and the modeled process. The state can change by firing a transition, which consumes a token from every input place of the transition and produces a token in all of its output places.

With this basic concept, it is possible to model very complex process logic, e.g. procedures in a logistics company. Because of the simple graph structure of Petri nets, it is possible to submit a model to in-depth mathematical analysis of its properties. This designates them to be a useful tool for the professional definition of complex processes as shown by Salimifard and Wright [64]. The general applicability of Petri nets for workflow management was shown by van der Aalst [53].

## ***Unified Modeling Language***

The Unified Modeling Language (UML) was introduced in the 1990s in order to combine common modeling notations in the context of object oriented programming. The Object Management Group (OMG) started to maintain UML in 1997. Its current version is 2.4 and it has become the de facto standard for modeling of software systems [62].

UML is specified by one extensive meta model called superstructure, whose different units allow flexible modeling of both structural and behavioral patterns. Processes can be represented by different kinds of models, e.g. activity diagrams, interaction diagrams, and state diagrams [54, Chap. 2].

Interaction diagrams are used to model different participants in a process and their interaction over time. This is contrary to the necessities for modeling a pipeline that focuses on the succession of jobs and therefore this class of UML diagrams seems not ideal for the purpose at hand.

State diagrams are the graphical representation of a finite-state machine. Thus, they consist of a number of states and the transitions between them. They do not explicitly model activities, which are the most important parts in a pipeline focusing on various steps and their execution semantics.

Activity diagrams are the attempt to create a modeling technique for this purpose, with special regard to business processes. Accordingly, they support not only the explicit modeling of activities, but also data objects, different roles and other notations used in business process modeling [60, Chap. 3].



Russell et al. conclude that for most aspects of business process modeling activity diagrams are appropriate [63]. Although they name several shortcomings in modeling resources and organizational aspects, it seems that it is sufficient for the modeling of genome data processing pipelines.

## *Event-driven Process Chain*

The Event-driven Process Chain (EPC) is a semiformal modeling notation developed 1992 by August-Wilhelm Scheer [65, Chap. 4.3]. It is an integral part of the Architecture of Integrated Information Systems (ARIS), which aims at defining a holistic modeling approach for business information systems.

The main elements of an EPC are:

- **Events**, which represent the state of a process,
- **Functions**, which represent a task or activity, and
- **Logic connectors**.

In the flow of a process, events and functions should alternate so every action results in a new state. Information objects can be used to model data flow on an abstract level and organization units can be associated to events.

EPCs are frequently used in Small and Medium-sized Enterprises (SMEs) in Germany. However, they are not adopted by an organization like the OMG.

## *Business Process Model and Notation*

The Business Process Management Initiative (BPMI) introduced the Business Process Model and Notation (BPMN) standard in 2004. Since 2006 it is an official standard of the OMG, which released BPMN version 2.0 in 2011. Four main classes of elements exist in BPMN [65, Chap. 4.7]:

- **Flow objects** represent real world entities, such as activities, events, and gateways,
- **Artifacts** are used to model additional information, e.g. data objects, groups, or annotations,
- **Connecting objects** are any sort of line or arrow, which connect elements with each other, and
- **Swimlanes** are used to structure the responsibilities for parts of a process.

The actual work of a BPMN process is done in activity elements. They represent an atomic task that can be executed by a human or by a computer system. The logic of a BPMN flow is determined by gateways. The most basic ones are the parallel gateways representing a logical AND and exclusive gateways representing a logical



XOR. Every BPMN process needs a starting event and at least one end event to mark the start and end of the process flow [65, Chap. 4.7].

There are further concepts in BPMN. Those, which are relevant for the execution of GDP pipelines are introduced in Section 2.3.2.

BPMN is widely used for the explicit modeling of business processes and it is well suited for integration tasks performed by a computer system. A complete formal execution semantic for BPMN is still in development [67, 55]. A number of workflow management systems already offer the automated or semi-automated execution of business processes. The applicability of these systems for clinical use has already been shown [58]. The Workflow Management Coalition (WFMC) was founded in 1993 to collect and standardize workflow related activities.

A serialization for BPMN can be found in the XML Process Definition Language. XPDL is a language for the machine-readable description of processes based on XML. It is a WFMC standard and designed to support direct storage of all elements of a BPMN model.

## ***Further Approaches***

Different genome browsers offer intuitive visualization of genome data. Many offer simple analyses to be executed on the data, e.g. to find interesting elements in a set of genome data. Examples for this are the tools offered by the National Center for Biotechnology Information (NCBI) described by Wheeler et al. or the ENSEMBL project [56, 66].

However, researchers usually need more detailed analysis features for their own data. The idea of offering them a possibility for these analyses without the need of programming skills is pursued by the Galaxy project as described by Goecks et al. [57]. Galaxy is an open source project developed by the Pennsylvania State University, the Emory University, and an active community.

The concept for the use of Galaxy is based on so-called tool elements. A tool represents an individual analysis that can be executed on a set of input data. Tools can be configured and executed by the user. To carry out more complex operations, it is possible to arrange several tools in a sequence to form a workflow. Workflows are designed in a custom editor provided by the project software. They are stored in the system and can be reused in future projects.

## **2.3 Modeling of Genome Data Processing Pipelines**

In this chapter, I describe the basic requirements for the modeling of Genome Data Processing (GDP) pipelines. The suitability of different process modeling notations is assessed and a modeling schema for these pipelines in BPMN is introduced. Furthermore, I describe how these design elements are stored in XPDL for the ex-



change between different parties. I selected BPMN for modeling because of its suitable collection of objects as well as its usability and intuitivity. XPDL was chosen for persistence as it is an established standard format for BPMN models.

### **2.3.1 Requirements Engineering**

In this section, I define the requirements of a modeling notation for GDP pipelines. I describe the elements needed to model the process flow and outline the concept of pipeline models and their instances.

#### **Modeling of Execution Information for Pipelines**

The smallest unit to be referenced in a genome data processing pipeline is a job. This term refers to an actual script executed on the server to perform a specific task while activity describes the abstract representation of a job in a process model. Thus, the most fundamental precondition for modeling of GDP pipelines is a representation of a number of jobs and their execution sequence.

Often a number of jobs can be seen as a logically associated group with one discernible purpose. In most cases these composite jobs can be used in several pipelines. To enable the reuse of composite jobs and provide an additional benefit of modeling, an implementation of a modeling system for these pipelines should support the definition and inclusion of sub pipelines. In every pipeline, it should be possible to call another pipeline instead of a job. This provides users with the possibility to arrange a pipeline in a hierarchical fashion, which removes the necessity to model a sequence several times, and reduces the complexity of models.

For the efficient execution of an alignment pipeline, it is essential that parallel data processing is used whenever possible. This has to include parallel execution of different jobs as well as multiple instances of one job or sub process.

There are jobs that have a varying internal behavior or outcome depending on input parameters. For example, an alignment behavior might support a dynamic parameter for the reference genome that is to be used for comparison. The parameter would be specified during construction of a pipeline using that job. This approach could reduce redundancies in different job definitions. For this to work there must be an option to model input parameters for the jobs of a pipeline.

#### **Model and Instance of a Pipeline**

In order to exploit the whole range of possibilities that arise when using a model-based workflow system, it is useful to distinguish between the model and the instance of a process pipeline. This distinction is necessary when a model contains variable parts. Without the specification of such parts, a pipeline cannot be exe-



cuted and different values for the variables can result in significantly different execution semantics. Hence a model with variables represents a group of pipelines or parts of them, comparable to a blueprint. In order to create an executable pipeline, the variables have to be defined. The model and instance of a pipeline further need to be distinguished from the process instance, which represents the actual execution of a pipeline instance. It should be possible to start with different input data in every run of a pipeline instance.

While the introduction of modeling aims to reduce the complexity and effort to create a new pipeline, the main purpose of variables is to enable the reuse of models. In the context of GDP pipelines a main process contains activities that can be done by a variety of different algorithms. Only the main pipeline has to be modeled once with a placeholder for these parts. For example, if in a pipeline various algorithms can do a specific task, only a generic term for the type of algorithm would be inserted before letting the user choose between all available algorithms of this type. A precondition for the usability of such a system is the existence of a classification of pipelines. This system has to enable the user to insert a name for a class of pipeline models that can be used for the calculation of the results for this step.

Similar to sub processes, parameters only show their full benefit, when they can be set dynamically. Some parameters can have fundamental impact on the execution semantics and on the result of a pipeline, e.g. the selected reference genome. Consequently, different values for the parameter set of a pipeline model should be accounted for as the definition of a new pipeline instance, not merely as different input parameters for the process execution.

Both sub processes and job parameters need to be available in a static and a fixed version. Thus, an implementation of such a system needs to find a way to label these spots. It also needs to provide an easy to use UI for the conversion of a set of models into a pipeline instance.

## Formal Requirements

Apart from the possibility to model the concepts described in the previous sections, in my opinion, a technique for modeling of GDP pipelines has to fulfill the following criteria:

- Intuitive graphical notation,
- Prevalence of the notation, and
- Standardized machine readable representation.

The subject of the intuitive usability of a notation is heavily dependent on the context and its intended use. In this case, the criterion is that the described pipeline elements are not only supported but can be modeled directly in a simple unambiguous way that matches the intended use. For example, forks of the data flow should be obvious at a glance and parameters for jobs should be modeled in a way that directly indicates their function. This is important because it cannot be assumed



that all people professionally working with genome analysis are familiar with the specifics of all modeling notations. Every element with a complex representation in the model increases the risk of errors and reduces the usability of the system.

As the aim of this work is to create a system that allows different people to execute pipeline models on a central server, it is necessary that the modeling language has got a certain degree of popularity. An important criterion for this is the existence of advanced modeling tools for the selected language. Additionally, a standardized machine-readable representation of the graphical notation is essential to assure correct interpretation of models when sharing them between different institutions.

2.3.2 Modeling of Execution Semantics

According to the requirements specified above, it is possible to assess the usability of the different modeling languages for the activities at hand. Table 2.1 summarizes the key attributes for all of the notations introduced previously.

Name	Purpose	Std	GDPP
Petri Nets	Mathematically verifiable sequence modeling	✗	✗
EPC	Business processes	✗	✗
UML	Technical specification of software characteristics	✓	✓
BPMN	Detailed modeling of business processes and workflows	✓	✓

Table 2.1: Summary of evaluated modeling notations and their applicability for GDP pipeline modeling (Std = Standardized, GDPP = GDP pipelines, ✗= No or insufficient support, ✓= Full support).

With the definition of flow and data elements that I presume necessary for the modeling of GDP pipelines, Petri nets and EPCs can be ruled out as conclusive modeling techniques. Both have insufficient support for parallel execution of activities with a variable number of instances. Petri nets also do not provide methods for the modeling of data objects and sub processes. Additionally, both notations lack a standardized machine-readable representation that would enable sharing of models between different institutions and EPCs are not widely used except in Germany.

In contrast to the previous examples, BPMN and UML activity diagrams support all the elements described previously. They both have at least one standardized XML representation. The graphical and logical elements are very similar to each other. A distinction can be made by the general orientation of the techniques. While activity diagrams aim for the extensive specification of software flow, BPMN focuses on the modeling of automated and manual business processes and intuitive usability by non-experts.



The question, which of these techniques is more fitting for different use cases is subject to numerous scientific papers [59]. The decision depends strongly on the specific situation. For the activity at hand, the main focus should lie on easy modeling of key elements. These are parallel activities or sub processes and input parameters for jobs. For the latter, there is a difference in the way of modeling. In activity diagrams, data objects are part of the process flow and usually simultaneously output of one activity and input for another. In contrast, BPMN shows input and output data as data objects associated with one or several activities independent from the process flow. This offers the possibility to store input parameters as data objects that are input parameters of an activity without being the output of a previous activity.

With these differences in mind, BPMN has two advantages: a more fitting concept for the representation of data objects for this case and the benefit of being developed with special focus on usability and intuitive modeling elements. For this reason, I chose BPMN as modeling language in the remainder of my work although UML activity diagrams might also be a promising candidate.

## Process Flow

Every job to be executed in the pipeline is represented by an activity element in BPMN. Although a job might have several predecessors or successors, every activity should only have one incoming and one outgoing flow in BPMN. Every process pipeline should have one starting and one end event. Figure 2.1 shows a basic form of a pipeline.

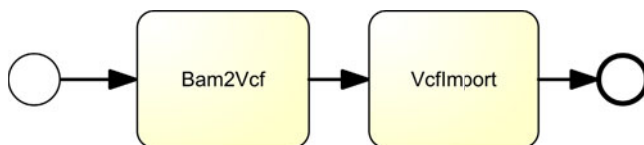


Fig. 2.1: Basic principle of a genome data processing pipeline modeled in BPMN.

## Hierarchy of Jobs

The processing pipelines can be nested hierarchically to any depth desired. Any set of logically associated jobs can be represented as a separate file containing a process diagram. In the invoking process a sub process activity is used as a placeholder for the nested jobs. The name of that activity has to be the same as the name of the sub process in order to automatically insert it in the parsing process. Figure 2.2 shows the usual depiction of a sub process in BPMN.





Fig. 2.2: Modeling of a sub process.

## Parallel Execution of Jobs

There are two ways of defining the parallel execution of one or several jobs. The first one is using parallel gateways. When a parallel gateway is signaled, all outgoing edges of the gateway are signaled. When used with one incoming and several outgoing edges, the gateway only signals after all incoming edges are activated. So the sequence flow can be split in two or more parallel strands and re-synchronized if needed. An example for this is shown in Figure 2.3.

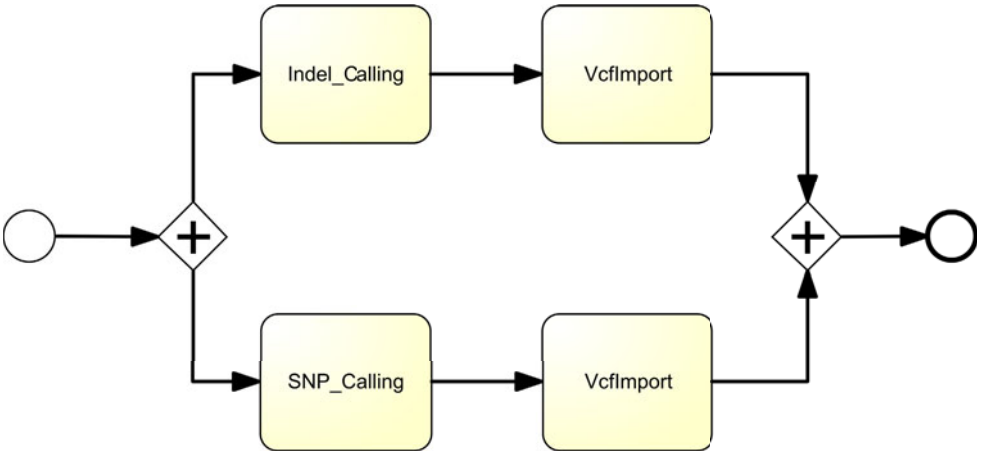


Fig. 2.3: Modeling of parallel gateways.

Often it is necessary to execute one job simultaneously several times. In that case it is preferable to use a parallel multiple instance activity, which is depicted as an activity with three vertical lines at the bottom as shown in Figure 2.4. In the BPMN standard, there are several ways to define the number of instances of a multiple instance activity [61, Sect. 13.2].

These possibilities are implemented in XPDL differently, depending on the modeling tool used. Additionally, most tools only allow the insertion of numeric values in the according fields. This is problematic when modeling a sub process with a variable number of instances as described in Section 2.3.2.

In my system the number of job instances to be started is included in square brackets behind the name of the activity. This circumvents problems arising from



High-Performance In-Memory Genome Data Analysis  
How In-Memory Database Technology Accelerates  
Personalized Medicine

Plattner, H.; Schapranow, M.-P. (Eds.)

2014, XXI, 223 p. 78 illus., Hardcover

ISBN: 978-3-319-03034-0