11

Control- Versus Data-Driven Workflows

Matthew Shields

11.1 Introduction

Workflow is typically defined as a sequence of operations or tasks needed to manage a business process or computational activity (Chapter 1). The representation of the sequence of operations or tasks is handled in many different ways by different people and varies from simple scripting languages, through graphs represented in textual or graphical form, to mathematical representations such as Petri Nets (Chapter 13) or $\pi$-calculus (Chapter 15).

Most groups agree that there are two simple classes of workflow structure into which most of the representations of workflow languages used in this book fall: control and data flows. The two classes are similar in that they specify the interaction between individual activities within the group that comprise the workflow, but they differ in their methods of implementing that interaction.

In control-driven workflows, or control flows, the connections between the activities in a workflow represent a transfer of control from the preceding task to the one that follows. This includes control structures such as sequences, conditionals, and iterations. Data-driven workflows, or data flows, are designed mostly to support data-driven applications. The dependencies represent the flow of data between workflow activities from data producer to data consumer.

There is also a smaller set of hybrid workflow representations based on a combination of control and data flows. These hybrids use both types of dependencies as appropriate but are normally biased toward either data flow or control flow, using the other to better handle certain conditions. For instance, in a data-flow system such as Triana (Chapter 20), there are situations where a downstream task needs to be activated but the upstream task produces no output. In this case, a \textit{trigger} is used to switch the flow of control. In hybrid control-flow systems, such as the CoG Kit’s Karajan workflow (Chapter 21), data dependencies can be represented by a \textit{future}, the concept of data that has not yet been produced, which can block the control flow with a data flow.

This chapter will examine the differences, and indeed similarities, between control flow, data flow, and hybrid representations, with examples of each type.
and the applications and frameworks that use them. We will start with a dis-

cussion of different workflow representations and some common concepts and

conclude with some of the pitfalls and some possible solutions to the problems

associated with heterogeneous workflow languages in Grid environments.

11.2 Workflow Representations

The data-driven versus control-driven workflow argument has run for as long

as workflow techniques have been in use and can be as evangelical as the

choice of editor, Vi or Emacs, or programming language, C++ or Java. Both

sides are convinced that the structure they use is the correct one, but there

are cases for the use of both workflow representations, and as we edge toward

interoperability and a common workflow language, mixed usage. The choice

of which is used in any given framework usually comes down to the original

application domain that drove the framework development, as we will see

when we examine some examples.

11.2.1 Common Workflow Terminology

It is worth mentioning here some of the common workflow terminology that

gets used within the various representations and frameworks. Workflow by its

definition has a number of common concepts; however, these are often known

by different names.

By definition, a workflow is a sequence of operations or tasks needed to

manage a computational activity. These are typically represented graphically

as a node on a graph or in a script as a process or a job. In Chapter 12, the

author describes component and service-based workflows, so we also have the

terms components and services used as a name for the computational processes

in the workflow. Different workflow frameworks also have different names for

this concept: in Kepler (Chapter 7) they are called actors; in Petri Net theory

(Chapter 13), transitions; in Virtual Data Language (VDL) (Chapter 17), pro-
cedures; in Cactus (Chapter 25), thorns; in Askalon (Chapter 27), activities; in

the CoG Kit’s Karajan (Chapter 21), elements; and in Triana (Chapter 20),

units. Although all of these terms hide different mechanisms and technolo-
gies, the basic concept of an operation or task holds, and we can think of each

of these as a “black box” process that performs some computation or other

operation.

The connections between operations are also known by different names:

vertices in a graph, edges in Petri Nets, pipes in data-flow systems, and mes-
sages in service-based systems. They all, however, represent an order to the

execution of the operations in the workflow. This order may be a data de-
pendency, where the product of the first operation in a connection must be

available for the execution of the second operation to start, or a control de-
pendency, where the flow of execution passes from the first operation to the