9.1 Introduction

Runtime, with regards to the YAWL environment, refers to the period when the YAWL System is active and executing – the host server is operational and the Engine is running in its servlet container (Apache Tomcat by default), accepting requests from Custom Services and applications to load specifications, start process instances (or cases), check out work items, and so on, generating events and progressing cases as per their control-flow towards conclusion. While much of what occurs during runtime is described in other chapters, most of the internal operations have been hidden. This chapter emphasizes the internal execution mechanisms of the YAWL Engine.

9.2 Basic Operations

The primary responsibility of the Engine is to execute process instances. However, there is a distinct demarcation between the work performed by the Engine and that handled by custom services. Throughout the life-cycle of a process instance, the Engine will prepare work items (i.e., task instances) for execution, at the appropriate times according to the specified control-flow of the process. But, it is important to realize that:

- **The Engine is not responsible for how the work of a task instance is performed.** Each and every task is associated at design time with a chosen Custom Service that will be responsible for performing the work of the task instance. If the association of a task with a Custom Service is not explicitly specified at design time, the Resource Service is associated implicitly as a default. The Engine announces the enablement of each task to the specified Custom Service at the designated
time with regards to the control-flow of the process, then awaits further requests from the service as to obtaining ownership of the task instance (via a “check-out” call) and later passing ownership back to the Engine (via a “check-in” call); what happens in relation to the task in the meantime is of no concern to the Engine. At check-in, the Engine will validate the data values assigned to the task instance by the Custom Service against the data schema of the task to ensure that the data is valid before the check-in is accepted, but it is not concerned with how that data was actually assigned to the task instance by the Custom Service (e.g., through user input, database lookup, web service response values, and so on).

- **The Engine knows nothing about users**, but only of its registered custom services. It may be that a Custom Service manages a set of users (e.g., the Resource Service) but the Engine allows only custom services to connect to it. Thus, it is said to be completely agnostic to “physical” users.

- **The Engine is unconcerned about how a task is resourced.** This is a corollary to the second point above: the Engine will pass responsibility for a task instance to a Custom Service, but has no concern for which “physical” user actually performs the work of the task instance, if any. In other words, the entire resource perspective is handled externally to the Engine and wholly within the Resource Service.

The two perspectives that the Engine is aware of and manages for each process are the control-flow perspective (determining which task(s) are enabled at certain times during the life-cycle of a process, based on arcs, conditions, splits, joins, and so on), and the data perspective (mapping data values to and from tasks and their parent nets, performing transformations, and evaluating expressions using the specified XPath and XQuery predicates). How it manages these two perspectives at runtime is detailed later in this chapter.

The Engine can execute a number of process instances concurrently, each one an instance of a **process specification**. A specification is expressed as an XML document, typically stored in a disk file, that describes the structure, format, data, and layout of a process that has been expressed graphically in the Editor. An XML specification file is produced when the process is saved in the Editor (the layout information stored is used by the Editor when a specification file is reopened, but is ignored by the Engine). Each specification is loaded into the Engine via Interface A – the Resource Service provides a web form that allows for the loading of specifications (see Fig. 9.1). Once loaded, the Engine stores the specification until such time it is manually unloaded (which will succeed only if there are no process instances based on it currently running).

Several versions of the same specification can coexist in the Engine at the same time (i.e., having the same specification identifier but different version numbers). Figure 9.1 shows two versions of the Order Fulfillment specification loaded and running, for example. This allows for currently running process instances to continue to completion, even though a newer version of the specification is loaded into the Engine; however, new case instances may only be started from the most recent version loaded.