# Guide to Orchestration in SmartFrog

Last Updated: 08/04/08

Contacts: [andrew.farrell@hp.com](mailto:andrew.farrell@hp.com), [paul.murray@hp.com](mailto:paul.murray@hp.com), [patrick.goldsack@hp.com](mailto:patrick.goldsack@hp.com)

## Introduction

The purpose of the orchestration extensions is to provide a means by which the execution of logic within SmartFrog components may be orchestrated.

The fundamental concept is as follows. An orchestration is a set of SmartFrog components. Each component in the set defines dependencies which guard whether it may change state.

As a trivial example, consider an orchestration consisting of three simple managed entities. Each managed entity may be created and subsequently removed. The initial state for each entity is that it is neither created nor removed. Consider also that the second (resp. third) may not be created until the first (resp. second) has been created. Conversely, the second (resp. first) may not be removed until the third (resp. second) has been removed.

This orchestration is depicted at least to some extent in the figure.

In the figure, there is for instance a dependency on the second managed entity that the first has been created. Not specified in the figure, but necessary would be the fact that the dependency is only relevant if the second has not been created already.

Notably, in the current approach we define dependencies for a component, but we do not define which states, or values for its attributes, a component may move to, or assume. This is really an omission and will be clarified in evolutions of this work. So, in the example, we should say that when first’s created attribute is true and second is not yet created, second may set its created attribute true. Currently, the component will be asked to do something, but what is not specified. (In fact, it is asked whether it wants to do something in response to being enabled, and if so told to do whatever it wants to do).

In order to achieve the desired functionality for this particular orchestration (as previously described), we must hard code certain aspects in the logic for the component. As said, in future a lot of what is hard coded will move into the orchestration model. In the following section, we present the model and the component logic which effects this orchestration. Then, we provide further details of the modelling approach and implementation.