Motivation

Clayton Christensen [1-2], as shown in figure 0.1, has the following view of theory building: To the left is empirical work leading to a descriptive theory and to the right is formal work leading to a prescriptive theory. In this approach, it is important to recognize the importance of descriptive empirical work. According to Christensen [1-2], when researchers move through the left-side funnel through observation, categorization and association – and in doing so develop constructs, frameworks and models – they have followed the inductive portion of the theory building cycle; when researchers move through the right-side funnel through hypothesize, test and validate – and in doing so prescribe constructs, frameworks and models – they have followed the deductive portion of the theory building cycle. As practitioners with some experience in building systems, the “inductive” path starting from the left-side funnel is adopted in this thesis for delving empirically and experimentally into Software Plug and Play.

Is Architecture an artefact? If so, what is it? Except in the architecture of buildings, wherein architectural drawings are the artefacts produced by a professional architect, this does not seem be apparent in the case of software architecture. One may contest this observation by claiming that Architecture description languages, have been devised explicitly for this purpose. However, traceability from Architecture to Engineering to Construction is arbitrary in conception and execution. The standpoint
in this thesis is that, software architecture is an abstraction of software, within the confines of which engineering the software artefact is manifest. Christopher Alexander’s seminal work on Pattern Languages [3], not just Patterns, is an attempt to characterize such abstractions.

In the present study, software architectures as instances of the notion of Plug and Play, has been explored and experiments have been performed to create pure Plug and Play Systems, that is, components, at different levels of abstractions, were created and composed purely by using only Plug and Play ideas. The semantics of the components were defined by methods, as is done to specify operationally the dynamics of objects. Such compositional Systems were visually created by drag and drop and connect operations on a grid, and operationalized through an execution or a simulation primitive defined over the composition. Six such experiments validated the feasibility of Plug and Play for composing software systems. The principal motivation for this exercise is to identify Plug and Play architectural abstractions so that they can be used to design Plug and Play based software systems. In practice, these are very critical to proper factoring and polymorphism in all vertical markets. To this end, it became important to capture this notion at the architecture level and started the whole journey.