Chapter 1: Introduction

To start with, the thesis author would like to state his position on Plug and Play. Plug and Play has been a useful architectural abstraction in Hardware. With the addition of Plug and Play capabilities to the operating system that drives the hardware, it became easier to setup, configure, and add hardware devices to a computer. As a result, resource rich, pervasive, computing and communication infrastructures have become the norm of the industry. In hardware, Plug and Play enables discovery and control of a wide range of devices over a breadth of categories while leveraging a standardized infrastructure. This has opened up new possibilities for hardware, as computing and communication power is being added to smaller, more common devices. As a result of which, everyday tasks are easier to accomplish [4] [5] and it has become easier to customize computers to specific needs of end-user.

This thesis proposes Plug and Play as a useful architectural abstraction in Software. This is necessitated by the need of creating extensible software systems. Even though design patterns and component frameworks are used in developing extensible systems, there is too much variety in these patterns and frameworks as a result of which software architects often use templates and adhoc techniques which make software systems built based on these patterns and frameworks code-centric, fragile and error prone. This thesis argues that the difficulties in the development of extensible software systems are due to the lack of appropriate architectural abstractions and proposes Software Plug and Play as a useful architectural abstraction for enabling extensible software systems. Software Plug and Play is driven by the promise of extending software during run-time by plugging in off-the-shelf compatible components in such a way that the plugged in components automatically adapt themselves to satisfy the needs of the software of interest. Furthermore, this reconfiguration of software components takes full advantage of the capabilities that are available from all the components (plugged in and existing).

In this chapter, the rationale for this proposal, and a definition of the Plug and Play abstraction is discussed. Further, an empirical approach that serves as the basis for establishing this rationale and the Plug and Play abstractions is put forth.

1.1 Why Plug and Play?

1.1.1 Case 1 – Supporting Change in Information Systems

Information systems have important strategic impact on global business which is evident in variety of industrial sectors like retail, manufacturing, services, healthcare, insurance, telecom and government. In
these sectors, every organization is a patchwork of information systems providing functionality in some identified and chosen aspect of business. Each system internally house-keeps transactions, supports processes and hosts business decisions to realize the functionality. They provide a platform that enables organizations to integrate and coordinate their business processes and ensure that information is shared across all functional levels and management hierarchies in the organization. In order to support change in the environment in which businesses exist, organizations constantly request for change in their supporting infrastructure which is percolated as changes to be made in the supporting information systems in terms of the functionality that they provide, business processes they support and business decisions that they host.

Currently, in order to facilitate change in the business environment, organizations go from a previously acceptable information system to an evolved version of an acceptable information system that correlates to the change in functionality, business processes and business decisions (in other words, change in requirements) needed by the organization. This change can be effected either by changing the configuration of the underlying components and their relationships or modifying existing Components to cater to the change or building new components and establish interrelationships between them in order to satisfy the change. In essence, the change in the information system is effected by following a Reverse Engineering Process till the causes for change in the information system are manifest, followed by a Forward Engineering Process in order to produce the new information system that satisfies the changed requirements. After this, the old information system is brought down and the new system put in its place and brought up so that the desired changes can be supported. It is evident that there is considerable business latency from the conception of change to its implementation. Such a situation is not desirable for businesses.

Business demand hot-swappable information systems by means of which parts of an information system can be replaced on the fly while the system is running and changes in information systems can be made with negligible or no latency. It is not hard to see that propagating changes with minimal latency increases value delivered by a business. To support this, it is proposed to introduce Plug and Play as a means to reduce/remove business latency impended in maintaining information systems thereby supporting hot-swappable information systems. Towards this end, Plug and Play software architecture, has been proposed, wherein information systems are built using Components that are composed on the fly during run-time. When all the components that need to be replaced due to change in requirements are composable during run-time, then the entire system becomes Plug-and-Playable. Such an approach
reduces latency, improves modularity, increases reusability, increases efficiency and reduces burden imposed by current processes.

1.1.2 Case 2 – Supporting Realization of Business Components

In many of the industry sectors, businesses utilize business components and associated business processes to perform their operations [6] [7]. They use information systems to support their internal processes as well as their customers in utilizing the different business components. Their marketing division studies the global market and identifies new business components that need to be supported by their information systems which are then implemented over a time frame. Based on the complexity of the new business component and the extent of its impact on the existing processes and other dependencies, it takes considerable time for their IT infrastructure team to develop and deploy the business component in the software world. It is not hard to see that there is latency between the conception of the business component and its availability for business use. This gap could very well dictate the success of the specific business component in the highly competitive global market. It is obvious that the organization that has the ability to realize a business component in a shorter time frame gains a higher market share as well as the first-mover advantage. Currently, if the underlying platform that an organization utilizes in implementing business components does not have the ability to turnover these components on demand then there is considerable latency involved in bringing out the business component which results in a make/break situation for the business. Towards this end, it is proposed to use Plug and Play software architecture wherein business components are plugs that are composed on the fly during run-time in such a way that organizations can perform businesses on the plugs and unplug them whenever they get outdated. When all the business components of the organization are composable during run-time then the entire business system becomes plug-and-playable. Such an approach reduces latency, improves time-to-market, gives first-movers-advantage, increases efficiency and reduces burden imposed by current practices.

1.1.3 Case 3 – Supporting Software applications Extensibility

The existing Plug and Play technology [8] [9] in Windows™, Macintosh™ and Linux operating systems has an underlying computer architecture that provides peer-to-peer connectivity of intelligent appliances, networks and wireless devices. This design brings in easy-to-use, zero-configuration, automatic discovery and release, flexible, standards-based connectivity to ad-hoc or unmanaged devices across a breadth of categories from a wide range of vendors. The industry view is that such Plug and Play architecture provides standardized access to computing resources and also has provisions for accessing future computing
resources (predominantly devices). This architecture has specifications for interfaces, buses and protocols as defined by different manufacturers based on which hardware devices can be inserted and removed, connected or disconnected without changing the configuration parameters. Software systems accessing these devices adjust to the new configuration automatically and do not need manual configuration file modification. It is proposed that this analogy is applied to Software, by which it is expected that if the features of all software applications in a computer System is Plug and Playable, then software applications would evolve as new features are plugged in. In a similar way, it is suggested that if, by design, the information structures maintained by different Software applications in a computer system map themselves to a common standard/specification then information can be exchanged with ease between these applications and information integrity maintained. Such an approach can increase productivity, improve reuse, improve extensibility, increase efficiency, and reduce time and effort expended by current practices in developing software systems.

1.1.4 Case 4 – Supporting Mergers and Acquisitions

Information systems are designed to support the functionality needed by business units so that they can meet their responsibilities. When business units are mature enough, they procure and/or home-grow and/or adopt best in class solutions available in market. (Example: Systems like ERP, SCM and CRM cater to different functional needs of business units but not to business as a whole). Ideally, these systems form the backbone of information flow by providing platforms to integrate and coordinate business processes and sharing information across all functional levels and management hierarchies. However, because of diversity of information systems procured over a period of time, there exist less connectivity and information structuring between these Systems. The impact of such inconsistencies is more when there are mergers and acquisitions, where in businesses are left with incompatible information systems and information structures. When two organizations merge, there is often an expectation that the back-office processes and systems supporting the two businesses merge onto a single platform. If the prior operating configurations of the two businesses are similar then there is a possibility that the supporting information systems infrastructure could be merged.

However, in practice it is never the case. Ideally, businesses expect a single platform supporting multiple businesses but they end up having multiple platforms supporting different businesses with each supporting different ways of running the business. Based on the size of the merger/acquisition, an informed choice needs to be made about migrating relevant information structures from one of the platforms to the other or build an altogether new platform in order to support the new business. In such
a scenario, Plug and Play of platforms can be a viable solution. In such cases, post-merger integration of two businesses could be made possible by plugging the two different platforms together as Components of a larger system. At present, this remains an idealistic goal of large industries which are looking at mergers and acquisitions as a way to grow and consolidate their businesses. It has the potential to disrupt current practices in maintaining enterprise information systems and operating businesses.

In the subsequent sections, a definition of software Plug and Play that clarifies the author’s position is arrived at. Before that, a literature survey of the existing definitions of Plug and Play is taken up so that the proposed definition can be seen in the context of past work. After that, a literature survey of the existing approaches to enable Plug and Play is taken up.

1.2 Review of Literature on definition of Plug and Play

According to Microsoft [10][11][12], Plug and Play is a combination of hardware and software support that enables a computing system to recognize and adapt to hardware configuration changes with little or no intervention by a user. It enables a user to add or remove devices from a computer system without having to manually configure its configuration switches or associated hardware. This facility offered by Microsoft requires support from device firmware, system software and associated drivers. Plug and Play is facilitated by industry standards which define the approach for automatic and dynamic recognition of installed hardware, hardware resource allocation and reallocation, loading of appropriate drivers, programming interface for drivers to interact with the device and finally, mechanisms for drivers and applications to interact with each other and perform appropriate actions. In essence, Plug and Play is a mechanism to simplify device driver development and device management so as to recognize and support hardware changes with very little user intervention.

According to the Universal Plug and Play Forum [4][5], Universal Plug and Play technology defines an architecture for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices and personal computers of all form factors. It is designed to bring ease of use, flexible, standards-based connectivity to adhoc and unmanaged networks. The Universal Plug and Play device architecture is more than just a simple extension of the Plug and Play peripheral model. It is an industry standard that is designed to support zero-configuration, invisible networking, and automatic discovery for a breadth of device categories from a wide range of vendors by utilizing common protocols that is media independent. It defines the protocols for communication between controllers (or control points) and devices. Devices that support universal Plug and Play can be implemented in any programming language and on any
operating system. It is an industry initiative that enables easy and robust connectivity and interoperability among devices from many different vendors.

While Plug and Play as a design pattern or Architecture is not supported in Eclipse [13], it supports the notion of Plug-ins which are pluggable components that add new processing elements to the existing system. In Eclipse, a plug-in is a component that provides a certain type of service within the context of the Eclipse workbench. It connects with a universe of other plug-ins to form a running application. A plug-in is an encapsulation of behaviour and/or data that interacts with other plugins to form a running program. Eclipse includes a plug-in management kernel (or) the Eclipse runtime that allocate/de-allocate necessary resources as well as activate/deactivate the plugin. The Eclipse runtime provides the necessary infrastructure to support the activation and operation of a collection of plugins that work together in order to provide this service. In essence, Eclipse plugins aid in achieving seamless integration and controlled openness by providing a common paradigm for the creation, management, and navigation of system resources.

According to the Intel specification [14] for Plug and Play, that was created in collaboration with Microsoft for the Windows™ Operating Systems, Plug and Play describes a series of specifications and features geared towards automatic configuration of devices. The objective is to make the devices and its corresponding software to work perfectly when first used or re-connected later without reconfiguration or adjustment by the end users. It describes the hardware specification that removes the need for user intervention in the configuration of peripheral devices. The various buses, devices and resources are abstracted from the base components via hardware-specific enumerators, arbitrators and device drivers there by providing extensibility in multiple operating environments. Plug and Play refers to the operating system functionality that supports discovery, addressing, connectivity, configuration and management of devices that are plugged into a computer system. In the context of the operating system, Plug and Play refers to the responsibilities and interfaces associated with the development of the appropriate device drivers that facilitates the Plug and Play functionality. In essence, Plug and Play as a technology allows for detection of devices without user intervention in discovery, connectivity and configurability.

According to Mezini and Lieberherr [15], Plug and Play is a software design that supports flexible white/black-box component composition thereby allowing incremental evolution of existing components collaborations, to create higher level collaborations out of simpler ones, while keeping the ingredients in the composition loosely coupled for better reuse. Such an approach produces and adapts software systems effectively by run-time composition of components. Plug and Play is supported by supporting
the primitive composition features of adding, deleting, modifying components during run-time; supporting component level variability; managing dynamic and flexible composition of components; providing configurable composition information; supporting expressive and changeable composition semantics and addressing dynamic reference problem of components. In essence, Plug and Play is a design philosophy that enables dynamic, flexible, hierarchical composition of components at run-time.

According to Wang et al [16], Plug and Play is an approach that allows software architects to experiment with alternative design choices of interaction semantics in a Plug and Play manner. This approach advocates that a library of pre-defined, reusable building blocks can be composed in a number of different ways to provide a wide range of component connectors with different interaction semantics. This is achieved by defining a set of standard interfaces that allow components to communicate with each other through different connectors. Such a Plug and Play approach allows designers to select specific interaction paradigms for component interactions, select specific connectors from pre-defined building blocks, and use finite state verification to evaluate the design. In essence, Wang et al, look at Plug and Play as a software design verification and validation approach rather than as a software architectural pattern.

According to David Berlind [17], APIs exposes service interfaces of potential software solution providers that conforms to certain specifications for consumption by potential users. Such APIs are designed to provide information (for example: content) or functionality (for example: location functionality) that is desired by the potential users. The software systems designed by the potential users consume the content provided by the invocation of these APIs and use them to provide meaningful information to their users. These systems can also invoke standard functionality provided by these APIs on content that is either provided by the software systems or obtained from other information sources. The use of APIs obfuscate the underlying complexities, internals and logic required to gather and organize content from multiple sources as well as to perform certain functionalities required in completing a real world task. By providing a library of APIs, it becomes possible to make complex processes reusable and also easier to connect to multiple systems at the same time. This in essence is a plug and play mechanism as outlined by David Berlind [17]. While APIs are indeed a useful mechanism to support plug and play, the level of abstraction is still at the programming level rather than at the architectural level. It is still a programmers’ approach of enabling extensibility of software systems and to improve their productivity in supporting content creation and building functionality. Moreover, the repository that contains the interfaces to the various APIs needs to be known beforehand at design time (even though the
orchestration and binding happens at run-time) for the various APIs to be invoked. This thesis is based on the principles that Plug and Play should not only provide orchestration and binding at run-time but also to provide discovery and invocation of APIs at run-time and all these should be defined as part of the software architecture as an architectural abstraction.

According to Thomas Erl [18], Service oriented Architecture (SoA) positions service as the primary entity for providing desired functionalities in a distributed computing environment. Each service exists as an independent software component that supports certain set of capabilities in a functional context. A collection of services from a service inventory is loosely coupled together to provide the desired functionality. To provide a new functionality that requires capabilities from different services, often it is necessary to create a new service that combines the necessary services and their capabilities at design time. Software systems built using SoA concepts, supports architecture extensibility in a few chosen dimensions that are identified during design time (For example: Any payment service, that supports the underlying standards and protocols, can be utilized to support the payment capability desired in a Retail system). In other words, the dimensions of extensibility is hard-coded into the architecture. Essentially, SoA based software systems find applicability in situations where specialized software (like retail solutions) that provide a variety of options based on pre-defined set of requirements are necessary. SoA based software systems typically use a layered architecture that represent key concerns that emerge in a solution. These layers can be categorized into implementation layer (addresses the interface with various services), consumption layer (addresses how the services are utilized) and information layer (addresses how information from/for services are managed and utilized).

### 1.3 Proposed definition of Plug and Play

As discussed in section 1.2, many practical definitions of Plug and Play exist. These definitions have evolved by the observation of design principles that Software architects adopt, norms that are agreed upon by the practitioners and actions architects perform while architecting software systems. Some consider Plug and Play as a Software Component that adds a specific feature to an existing software system so as to enhance the software system. Others look at it as an architectural design pattern that facilitates glass-box extensibility of software systems. Others consider it as an inherent capability of the software system to allow third-party developers to create abilities that extend the software system. Some others consider Plug and Play as a System design principle that facilitates future growth in certain chosen dimensions. Some others look at Plug and Play as a specification that facilitates discovery of pluggable Software Components in a System without the need for human intervention in configuration and
utilization. Some others feel that Plug and Play implies instant connection and instant operation immaterial of whether the components being plugged are hardware or Software. Some others feel that Plug and Play is a design philosophy in order to make computer hardware, the operating System that drives it and the drivers that connects the devices to the operating System to work together seamlessly without human intervention. Some others define Plug and Play as the ability to add a new Component to a System and have it work automatically without technical configuration. It is evident that there is no common consensus about these definitions, but each of these definitions has been practically found to be appropriate for a specific situation. With this background, a definition of Plug and Play is put forth.

Traditionally, ‘Plug and Play’ is considered as a collection of standards for Software and hardware, wherein Software refers to the computing resources in the broadest sense and hardware refers to the computing infrastructure that provides these computing resources, to support functional and structural extensibility of computing Systems that they are part of. While the term ‘Plug’ signifies instant connection and instant disconnection of the computing resources, the term ‘Play’ signifies instant operation and instant halt of the functionality that emerged due to the connection/disconnection of computing resources. The term ‘Plug and Play’ is the defining characteristic of the computing System as a whole that enables the user to enhance the computing System while it is running. Plug and Play can enable computing Systems to support specific extensibility, in a chosen dimension, or general purpose extensibility, in multiple dimensions. In order to apply this analogy to Software, it is necessary to map Plug and Play elements to corresponding software system elements. Since Components are the loci of computation and state, Components are the software system resources, the underlying Component framework is the necessary infrastructure that supports the necessary Composition model and the run-time provides the standards for Plug and Play of components within the constraints of the component framework.

This thesis proposes that ‘Software Plug and Play’ is a collection of standards, encoded in the run-time (which itself is a component) and embedded in the component schema, for Software Components and their Composition (supported by an underlying Component framework), in order to support functional and structural extensibility of a running software system. The term ‘Plug and Play’ signifies the defining characteristic of the run-time Composition (or decomposition) of the software system that enables users to enhance (or bring it to original state) the software system while it is running. The term ‘Plug’ signifies the defining characteristic of the components, which are to be composed together, in terms of their ability to be attached or detached during run-time. The term ‘Play’ signifies the defining
characteristic of the computation provided by the composed Components in terms of their ability to be
instantly invocated or instantly terminated after Composition or decomposition of the constituent
Components.

Towards understanding this particular viewpoint, a simple example of a bank offering net banking
System to its retail customers is considered. Depending on the nature of a customer’s business
relationship with the bank, some basic financial services (like savings and current accounts), and some
additional financial services (like demat services, fixed deposits, recurring deposits, credit cards,
insurance) are made available as part of the net banking infrastructure for that customer. While the basic
services provide minimal banking experience to the customer, the additional financial services help the
customer invest in different financial instruments with varying return on investments. Depending on the
returns that the customer obtains, the customer might invest in new financial instruments and drop
existing financial instruments or increase the investments in their current setup and this can occur at any
point of the financial year. There is also the scenario wherein the banks’ marketing division studies the
financial services market and identifies new financial instruments that their customer can additionally
invest in and makes them available as part of the net banking System. In order to support these changing
investment needs of the banks’ customer, the net banking System that is exposed to the customer should
be able to manipulate and deal with a multitude of financial instruments. Customers should also be able
to enrol into new financial instruments or drop existing financial investments. In this scenario, it is proposed that Plug and Play is a useful architectural abstraction for the net banking System.

Minimally, the components of the net banking Architecture are the financial Component framework that caters to the business practices and processes of the bank, the Plug and Play run-time that supports the financial Component framework and also enables manipulation of the different financial instruments and their associated services, the different financial instruments as Components that can be manipulated. Each of the financial instrument Components also has a minimal run-time that interacts with the component framework run-time and provides the necessary Plug and Play capabilities. As shown in Figure 1.1, at a higher level, the net banking System Architecture is composed of the financial Component framework (with an underlying Composition model), the run-time Component that enables the inner workings of the financial Component framework and the various financial instruments as pluggable Components.

1.4 Scope of this Thesis

This thesis investigates how the development of extensible, component-based software can be facilitated with software plug and play as an architecture abstraction. This problem is approached as programmers, experimentally, to be able to specify, to separate what from the how. Towards fostering clarity in this task, the thesis author proposes to specify architecture formally. To this end, the thesis author has undertaken several experiments, lessons from which potential formal aspects of specifications are identified. As said earlier, Plug and Play is potentially a useful architectural abstraction to solve many problems faced in the industry. Even though many attempts have been made to create Component models, techniques and frameworks that address some of the problems that Plug and Play can potentially solve, most of these efforts resulted in creation of silos that are incompatible and non-interoperable with each other. It is seen that the current interventions have compounded many small problems together to create variety of information system pathologies. In such a situation, treating Plug and Play as an architectural abstraction could be a viable solution that can address many of these pathologies.

Components exist in an environment in which they can be composed. Compositions create a relevant environment with respect to the problem on hand. Outside this environment is its run-time system, which makes it a truly operational entity. Such runtime systems are components too. They can be specified and implemented. Their implementation leads to an operational basis for perceiving a software architecture, even though this Architecture only specifies what the software system should do. In other words, software architecture is an abstraction. Its brush with reality stems from the fact that its runtime system is factually
implemented by hardware. In other words, software architecture is always based on presumptions of its implementable environment. For the purposes of this thesis, it is sufficient to note that this environment takes the shape of a component that specifies the constraints to be satisfied by its implementation. Going further, in order to address the Plug and Play problem in its entirety, the following needs to be considered:

a) Acquire empirical evidence that Plug and Play is a meaningful architectural abstraction. This is a necessity as while the notion of Plug and Play is understood by the industry, there is no common definition on consensus on what Plug and Play really is.

b) Derive the fundamental set of definitions of Plug and Play and other related entities. This is a necessity as there are numerous definitions of Architecture, Components and Composition and it is necessary to find the common thread among all of them.

c) Develop an approach to describe Plug and Play, Architecture, Composition and Components. This is a necessity as there are numerous ways of describing them and it is necessary to find a standardized approach to describe Plug and Play and related entities.

d) Apply the approach and the definitions to solve an industry problem (Mergers and Acquisitions or Features Plug and Play). This is a necessity as it serves as a good testing ground for the hypothesis and approaches on Plug and Play.

e) Utilize the applied research for working out the theoretical ramifications of Plug and Play. This is a necessity as the entire supporting environment for architectural design needs to be worked out.

In this thesis, the first 3 steps have been taken up and the rest is left as future work. The chapters in this thesis have been organized in order to cater to these 3 steps.

1.4.1 Empirical study of Plug and Play experimental Systems

According to Basili [19-22], empirical method of experimental design proposes a model, and studies the effects of the outcomes of adopting this model to develop systems. Basili [19-22] also discusses about the importance of validating the model by means of interpreting the results obtained by studying the underlying framework. In order to characterize the model in a realistic environment, Basili [19-22] proposes that multiple experiments supporting different configurations of the same model needs to be tried out. Accordingly, six graded systems with decreasing requirements expected of their environments have been architected, implemented and described in this thesis. These experiments are discussed in chapters 5 to 10. According to Kitchenham [23], the information that needs to be captured as part of the empirical study on software systems are: a) Context, b) Design, c) Analysis, d) Results, and e) its Interpretation. Accordingly, the problem and solution space, system requirements, solution quality
characteristics, architecture description of the system, analysis of the system and an abstraction of the analysis has been expressed as part of the discussion on experiments in chapters 5 to 10.

According to Ticky [24-25], repeating experiments under different real world situations is an essential ingredient of empirical work. Accordingly, in this thesis, each of the six experiments is conducted in a different domain under different environmental conditions and configurations as given below:

1. Experiment 1 is about architecting a management workbench for usage by managers in an information technology reliant environment. In this experiment, an entire management technique is treated as a pluggable component. The objective is to understand the fundamental constructs of plug and play. It is discussed in Chapter 5.

2. Experiment 2 is about architecting a systems modelling platform for usage by systems thinkers in order to capture their understanding of a system-of-interest. In this experiment, a generic modelling technique that can support different modelling languages and associated modelling notations is treated as a pluggable component. The objective is to understand the effect of multiplicity of plug and play components. It is discussed in Chapter 6.

3. Experiment 3 is about architecting a process modelling platform for usage by process designers in order to develop a process model. This experiment applies the plug and play concepts identified by experiment 2 in a different domain (systems domain to process domain). The objective is to repeat an experiment in a different context under different environmental conditions. It is discussed in Chapter 7.

4. Experiment 4 is about architecting a visual process programming platform for usage by organizations in order to express their operational processes as well as to support process execution by role playing agents. In this experiment, two different types of components (there can be multiple components but these components must belong to one of these types) are plugged and played in the platform. The objective is to understand how different types of plugs and corresponding components can co-exist and function together. It is discussed in Chapter 8.

5. Experiment 5 is about architecting a robotic process automation platform for usage by organizations to automate human tasks performed on a computer. In this experiment, a multitude of components that interact and automate tasks on different software technologies are dynamically plugged and utilized by the platform. The objective is to understand how to scale up the ability to plug and play components. It is discussed in Chapter 9.
6. Experiment is 6 is about architecting an instrumentation platform for usage by organizations to sense how different software applications are behaving and how users are using these software applications. In this experiment, a multitude of soft-sensors plug into different software applications and capture a wide variety of information. The objective is to understand how to handle the complexity of data streaming from multiple sources and to scaling up the operations. It is discussed in Chapter 10.

Based on the analysis of the architectures of these systems and their implementations, the plug and play architectural abstraction has been arrived at and discussed in Chapter 11.

1.4.2 Definition of Plug and Play abstractions

According to ISO 42010 [26], architecture is the fundamental concepts or properties of one or several systems in their intended environment embodied in their elements, relationships, and in the principles governing their design and evolution along their life-cycle. As discussed in section 1.2, there exists a wide variety of fundamental concepts that have been found useful in utilizing Plug and Play to architect systems. These concepts have evolved by the observation of design principles that software architects adopt, norms that are agreed upon by the practitioners and actions architects perform while architecting software systems. While Plug and Play is generally considered as an interface specification, this thesis considers Plug and Play as the defining characteristic of the software that enables the user to enhance the software while it is running.

Accordingly, this thesis considers, components (as they are entities that are plugged and played), their composition (as this the result of the plug and play operation), the resulting architecture (the structure that supports plug and play behaviour), plugs (as they are entities that enable plugging), sockets (as they are the entities that enable dynamic service invocation), run-time components (as they are the implementation of the plug and play capabilities) as the essential ingredients of the Plug and Play architectural abstraction. Accordingly, Chapters 2, 3, 4 discusses the architectural abstractions components, composition and architectures. Chapters 1 and 11 discusses about the architectural abstractions Plug, Socket and Plug & Play. Chapters 5 to 10 discusses about the different architectures and the related run-time components that have been found to practically useful.

1.4.3 Developing an approach to describe Plug and Play abstractions

The level of detail that needs to be provided in order to express a software architecture has been a difficult choice to make for software architects. Often, the choice of using a specific architecture
description technique and the extent of formalism of the architecture description has dictated the level and complexity of the details. The perception of the industry and the viewpoint taken by the software architects often dictate the level of details. On the one hand, architecture descriptions are treated as a work product emanating from an architecture process [27] while on the other it is treated as formal description of a system. Generally, the practitioners prefer to use something that is easier to understand, communicate, graphical and not necessarily formal, while software architecture researchers prefer to use formal syntax and semantics for expressing architecture descriptions so that they can move from such descriptions to code using tools. There have been many attempts by both industry [26] and academics [28-30] to bridge this gap by defining the essential ingredients for an architecture description; however, for various reasons all these attempts have failed in providing a definitive way to express architectures.

As a result, it becomes important for this thesis to identify an approach for expressing architectures as well as to identify the architectural elements that are expressed. There are a wide variety of elements (interface, process, components, composition, architecture, principles, guidelines, key properties, and so on) that are of relevance to the plug and play community. The eventual elements that were chosen were Components (as they are the fundamental structural elements of an architecture), Composition (as components needs to be composed together to create the whole from the parts) and Architecture (as architecture is a high-level structure) as they are sufficient in order to capture the plug and play architecture abstraction.

Since this thesis is about software architectures in general and plug and play in particular, it became necessary to choose a specific architecture description technique. There are a wide variety of techniques (UML/SysML views [31][32], Architecture models [26-27], Interface description languages [33], interchange description languages [34], architecture description languages [26], formal methods [35], and so on). One criteria that was used for selecting the technique is its ability to express components, composition and architecture; another criteria was its simplicity in expressing the chosen architectural elements and the last criteria was to apply a simple approach to validate the description. A simple syntax based on the architecture description template presented in Annex A, is used in this thesis to express the architecture descriptions of the different experiments. Chapters 2, 3, 4 discusses about the different architecture description approaches to express components, composition and architectures. Chapters 5 to 10 uses the template in Annex A to express the architecture descriptions of the different experiments. Chapter 11 uses this technique to express the plug and play architectural abstractions.