Chapter 7: Experiment 3 – Process Modelling Platform

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled Process Modelling Platform. While there exist a significant number of architectural elements and design patterns in this system’s architecture, the focus of this discussion is on those that are potentially useful as Plug and Play constructs. The purpose of this platform is to enable representation of a process using different process languages and associated diagramming notations. Accordingly, the platform adopts an extensible architecture so that new process modellers can be developed independently and then plugged into the platform on demand. The reasons for adopting an extensible architecture include catering to the complexity of the evolving adoption of processes in different industries and provisioning for extensions.

In the context of Plug and Play, the objective of this experimental system is to “adopt the plug and play based extensible architecture”, wherein the plugged components provide the ability to capture different levels of information. This is achieved by providing information gathering, manipulation and exchange services as part of the plugged components. Accordingly, the hypothesis for this experiment is: “If components are plug and playable, then irrespective of the domain, extensible systems become possible”. The core functionality of the platform is the ability to support a multitude of process languages and diagramming notations, and the ability to collect a variety of information pertaining to the encoded processes, which are implemented as corresponding modellers. The methodology adopted to validate the hypothesis is to treat each of these modellers as a pluggable component. Accordingly, the goal of this experimental system is to “enable Plug and Play of process modellers”.

This chapter is organized as follows: a) The section on the problem space provides an overview of the process modelling problem, b) The section on the solution space provides a possible solution that is Plug and Play compliant, c) The section on the requirements lists down the stakeholder concerns, d) The section on the quality characteristics lists down the functional and non-functional characteristics of the system, e) The section on decomposition brings forth the underlying component framework and supporting run-time, f) The next 4 sections provide an overview of the various components and their composition, and g) The last section discusses about the Plug and Play architectural abstractions of this experiment. It uses the architecture description template presented in Annex A to express the architecture description of the process modelling platform.
7.1 The Problem Space

A process can be recognized as a set of actions (either for people or for machines) performed in an organized manner under a set of constraints to achieve a particular goal [111-120]. It gives the ability to role players to discharge their responsibilities in a systematic structured way. Such, processes are encoded in such a way that the actions are exposed to the performing role players in a pre-defined sequence based on the desired outcomes. These actions are implemented and orchestrated seamlessly by these role players in order to achieve the desired process outcomes. Using formal representations to express processes helps organizations to understand, analyse and adopt processes in their various endeavours [113][117-118]. In practice, processes are represented as a hierarchical collection of process steps ordered towards achieving a desired outcome. Each process step has properties and parameters which detail the process step and provide relevant information that is useful for understanding, analysing and improving the process.

In practice, process modelling notations, like BPMN, Open Span, BPEL, IDEF, OPF, Flow charts [111-120] and so on, that are relevant to the purpose of the process are used to represent processes. Most of these notations, consider processes as a collection of tasks (or activities) that consume some input in order to produce some output while utilizing a set of resources and capabilities to deliver work products (or artefacts). Often, organizations utilize one or more process modelling notations to express their processes. These notations help organizations to visually represent the reality behind the processes and allow them to examine and understand the behaviour that will result in the desired process outcomes. Depending on the purpose for which the processes are put used to, it is necessary to use the relevant process modelling notation so that enabling technologies are available for operating on these processes. The different modelling notations allow organizations to deal with processes at different levels of detail [111-120] (BPEL is predominantly useful for process orchestration, Open Span is predominantly useful for process automation, IDEF is useful for conceptual modelling and so on). The existing process modelling tools are designed and developed by multiple vendors and support different process modelling notations and corresponding philosophies. Often, substantial effort needs to be put in to provide all the details required by most of these modelling tools. Process designers, often start with a bare essential process and iterate over it by repeated application in various engagements till it reaches a stage where it can be expressed in these tools. It is also the case that these designers use a rudimentary notation to express their processes in ways which may not be syntactically correct.
7.2 **The Solution Space**

Process design is considered as the act of putting together an integrated set of methods, practices and procedures in the form of a process whose execution can produce quality artefacts in a reliable, predictable and efficient manner [111-120]. Process models are considered as the work products used to express a process and process modelling is considered as the act of developing this representation by means of appropriate modelling conventions, diagramming notations, activities, conditions, constraints, resources, work products, inputs, outputs, outcomes and so on [111-120]. The act of process modelling starts with understanding the desired outcomes to be achieved by the execution of the process and the desired work products to be created by the process. The next step is to understand the inputs that are available for initiating the process and the initial conditions that trigger this process. Using this understanding, the process is expressed in terms of process steps, order of these steps, properties and parameters of the process steps, the desired outcomes that can be achieved by the execution of the process, and the resources that are necessary for executing the process steps.

Process models are usually expressed using an approximate symbolic representation by using a chosen modelling notation [111-120]. They represent the entities that are responsible for delivering the desired work products and the relationships between them. Process modelling results in the development of an analogical set of relations between actions, resources, artefacts and outcomes, which are collectively expressed as a model, catering to a predefined form and convention. While process design results in the synthesis of the process model, process modelling results in the elaboration and expression of the process model. The act of process design is iterative, by which details of the process are built by utilizing knowledge about how different process steps and the enabling resources contribute to the process outcomes and work products [111-120]. Hence, it is necessary to support the ability to dynamically change the process model based on the corresponding change to the design. This would involve changing the underlying information pertaining to each process step as well.

In this setting, the goals of the process modelling platform are:

- Create a modelling platform in which multiple modelling notations can be plugged in and can be instantiated in order to address the process designers’ needs.
- Support the development of a representation that is amenable for communication between a set of stakeholders.
- Provide a rigorous framework for estimating and planning the outcomes of the process and the ways and means of achieving the desired outcomes.
• Provide placeholders for metrics and measures that can be used during process and product quality assessments.
• Provide a rigorous framework for expressing the inputs, outputs, resources, artefacts, work products and constraints of the process and its constituent process steps.
• Support the four elements that are commonly found in process models: a) Notation, b) Structure, c) Rationale, and d) Expression.
• Support the four characteristics that are attributed to a typical process model: a) Order, b) Integrity, c) Consistency and d) Insight.
• Support process notation modelling and creation of corresponding process modelling tools.

7.3 **Requirements for the Process Modelling Platform**

i) Process designers should be able to select a specific modelling notation from a collection and express their process using this notation.

ii) Process designers should have the ability to modify the modelling notations according to their intended purpose of usage of the process.

iii) When the underlying modelling notation is changed, the process model should be able to accommodate the enhancements.

iv) Process designers should be able to express a process using multiple modelling notations.

v) Process designers should be able to exchange process steps and process fragments across different processes.

vi) Process designers should be able to work on multiple process models at the same time.

vii) Process designers should be able to utilize multiple modelling techniques to express a single system.

viii) Process designers should be able to expand the collection of modelling notations available at their disposal in the modelling platform.

ix) Process designers should be able to save the state of their models and resume modelling at a later point in time.

x) Process designers should be able to rework their models from any point of their model formulation.

xi) Process designers should be able to reuse their models across different modelling instances.

xii) There should be a common elemental data structure/schema for the models so that interoperability amongst the models can be supported easily. This data schema should be extensible so that new models can also be encoded (In other words, there should be a single meta-model).
xiii) There should be a common collection of primitive graphical user interface (GUI) elements using which
all the user interactions are expressed. This collection of GUI elements should be extensible so that
new models can be supported.

xiv) There should be a set of core functionalities that is common across all modelling techniques.
xv) Information pertaining to a model can be collected in any format. This information should be
converted and expressed in a unified format which is common across all models.

7.4 Quality Characteristics of Process Modelling Platform

Quality characteristics of a system are a set of essential and distinguishing attributes that have a
pragmatic interpretation of the system’s inferiority or superiority. By controlling the quality
characteristics, it is possible to ensure that the system delivers desired value to its stakeholders [100-102]
[132-133]. For this discussion purposes, the author utilized ISO 25010 [103] as the reference quality
model. The subsequent sections list down the quality characteristics of the Process modelling platform.

1. Functionality:
   a. Suitability: Aids process designers in expressing at least 80% of the processes that they
      encounter.
   b. Suitability: At least one modelling notation can be used by the process designer to model their
      process.
   c. Accurateness: Process designers should be able to accurately represent the model of their
      process along with sufficient detail.
   d. Interoperability: Process designers can transfer process steps and process fragments to
different modelling notations.
   e. Compliance: Complies with all the modelling constraints advocated by the modelling notation.
   f. Security: Process designers can be assured that the information expressed in the process
      model is confidential.
   g. Security: Uses open format for storing information in storage media and while exchanging
      information between models.

2. Reliability:
   a. Maturity: Modelling notations are mature enough to help address 80% of the modelling
      needs of process designers.
   b. Fault Tolerance: Any issues that arise due to implementation of a modelling notation should
      be handled gracefully.
3. Usability:
   a. Understand ability: Process designers can comprehend and use the modelling notation (100%)
   b. Learnability: Guides through the steps of using the modelling platform (50% guidance)
   c. Operability: Uses familiar interfaces for interacting with the User (80% compatibility)
   d. Operability: Undo/Redo last few actions performed by the user on the Platform

4. Efficiency:
   a. Time Behaviour: Process designers should be able to model their processes within a predefined time frame. The platform should not be processing forever.
   b. Resource Behaviour: Platform should use minimal resources to satisfy the modelling needs of the process designer.

5. Maintainability:
   a. Analysability: Process designers can keep track of progress always (100%)
   b. Changeability: Process designers can change their models and modelling notation on demand (100%)
   c. Stability: Absorbs 80% of perturbations introduced by the working environment.
   d. Testability: System performance and its use to model a process is reproducible (100%)

6. Portability:
   a. Adaptability: Adapts to change in the working environment (100%)
   b. Install ability: It should be easy to expand the collection of modelling notations that are available in the Platform (90%)
   c. Conformance: Conforms to Modelling constraints (100% compliance)
   d. Replace ability: Process designers can replace any of their Models and modelling notations (100%)

7.5 Decomposition of the Process Modelling Platform

Process modelling is the act of creating a representation of a process in a form that is suitable for communication to different users [111-120]. It reveals the gross structure of the process and allows one to reason about the process properties, process outcome properties, key performance indicators and process structure [111-120]. There are different approaches to develop the process models. They are: a)
Instantiating a meta-process (abstracted process model that is expressed using a set of concepts), b) Utilizing a language and associated notations to express the process (Little Jill, Prolog and so on), c) Assembling a set of components that are available in a process repository, and d) Formalizing experience in a domain. The first two approaches are widely prevalent and the platform is designed support these approaches as they provide a view of the whole process and requires the use of a notation (textual or graphical) or a language, which conforms to a meta-process-model, to express the process. Such notations or language comprise of a well-defined syntax and formally defined semantics that would help in some form of reasoning about the process.

Process modelling involves: a) Identifying the conventions that needs to be adopted, b) Identifying the modelling notation to be utilized for expressing the process, c) Identifying the activities/tasks/process steps that needs to be encoded, d) Identifying the inputs, outputs, conditions and constraints for each of these activities/tasks/process steps, e) Identifying the necessary human and organizational resources, f) Identifying the necessary organizational capabilities, g) Identifying the order in which these activities/tasks/process steps are performed, h) Expressing the process, associated information and their sequence in an appropriate form, i) Ensuring consistency of the representation, and j) Analysing the model.
for consistency with the actual process. The objective is to setup the necessary infrastructure that can aid in capturing all relevant information pertaining to the chosen convention for the process-of-interest.

Accordingly, the process modelling platform can be decomposed into the top level components: a) Process modelling framework that supports the ability to express different process designs, b) Process modelling framework run-time that supports the framework, c) Modeller container which manage a collection of modellers, the models that they support and the meta-models that they instantiate, and d) Model repository which hosts the different models that are manipulated by the platform. The decomposition of the platform is illustrated in Fig 7.1.

7.6 Process Modelling Framework

At the core of the process modelling platform is the process modelling framework (PMF) which provides a set of generic capabilities, resources and interfaces for process modelling. They serve as the foundation for the creation and assembly of process models. The core capabilities and resources are: a) Model rendering, b) Model storage and retrieval, c) Model transformation, d) Model traversal, e) Model assembly, f) Model extraction, and g) Model instantiation. In essence, PMF supports a set of conventions, principles and practices for modelling processes which can be used to represent processes using a multitude of process languages. Each process language and its underlying process meta-model is supported by the corresponding process modeller which is plugged into the PMF. The process platform can be used to create the process meta-models which can then be instantiated for developing the different process models.

In order to develop the process model, it is necessary to understand the process design and encode it using the chosen modelling language. Additional information like the inputs, outputs, constraints, work products, resources, artefacts, bindings and so on needs to be expressed using this language in order to have complete information about the process. In this endeavour, the process meta-model and the language that implements this process meta-model is of essence.

As shown in figure 7.2, PMF uses the services provided by the modeller to create a representation of the process that conforms to the specific process language. It uses the services provided by the model analyser to determine the characteristics of the process and the process outcomes. It uses the services provided by the model exchanger to transfer information between different modellers and to transform one view into another. It uses the services provided by the model extractor to obtain parts of a process model so that they can be either detailed further or used in a different instance. It uses the services
provided by the model assembler to string different process steps and process fragments together in the form of a process. It uses the services provided by the model renderer to generate different views of the process model. It uses the services provided by the model dashboard to provide information pertaining to the status of the different processes that are manipulated in the platform. It uses the modelling services provided by the plugged modellers to provide support for the different process languages and process meta-models. It uses the tree traversal services provided by the model traverser to navigate thru the process model hierarchy.

![Diagram of Process Modelling Framework](image)

**Figure 7.2: Decomposition of Process Modelling Framework**

### 7.7 Process Modelling Framework Run-Time

Process modelling often involves utilizing a variety of process languages and associated diagramming notations to capture a complete representation of the process. A modelling platform that supports all these different languages and notations is difficult to implement as a single monolithic system. Any
enhancement to one of the process languages or notations; or addition or one or modellers would result in a situation where the entire system needs to be repackaged and re-implemented, often resulting in upgrade nightmares. Modularity and extensibility are design principles that facilitate extension of the platform so that new process languages and associated modellers can be added to it as and when demanded. However, the class of extensions that can be supported at run-time are anticipated extensions. In the case of the process modelling platform, the two components that require variety management are the model analysers and the process modellers. In this architecture, only the variety in process modellers has been deal with and they serve as the extensions for the process modelling platform. A good design practice in this situation, is to embed the necessary extensibility services in the run-time system. The objective of the run-time system is to provide necessary interfaces that support interaction with the modellers, sharing of information with the modellers, integration of the modellers with the platform and management of model life-cycle.

![Diagram of Software Decomposition of Process Modeller Framework Runtime]

**Figure 7.3: Software Decomposition of Process Modeller Framework Runtime**

The bus design pattern, implemented as the model manager, can be used by the process modelling framework run-time [PMFR] for supporting information sharing between the modellers as well as with the model analyser, model renderer, model extractor and model assembler. The host-controller design pattern can be used to facilitate interaction between the modellers and the modelling platform. The
interoperability-bus design pattern can be used in PMFR for supporting information exchange between the modellers. The plug and socket design pattern can be used to dynamically plug and play process modellers at run-time. The event-driven design pattern, adopted by graphical user interface based operation systems can be used in PMFR to handle and process the various events that are triggered by both the platform as well as the modellers. The services design pattern, adopted by service-oriented-architectures can be used in PMFR to handle consumption and provision of services that are of interest to the other components of the process modelling platform. Finally, the data handling services provided by the data manager can be used by PMFR to supplement each process step in the process model with additional information as per the requirements of the process meta-model that the process instantiates.

As shown in figure 7.3, the modelling framework is a composition of: a) Modeller Socket which serves as the extension point for a modeller, b) Host Controller which serves as the node that provides resources, capabilities and services to the modellers, c) Information Bus which provides resources, capabilities and services for manipulating and transforming data, d) Event Handler which serves as the placeholder for invoking services/functionalities from various modellers and processing their results in response to the various events triggered by the modelling platform, e) Service manager which provides the mechanism for providing services that can be utilized by the other components and for consuming services that are provided by the other components, and f) Data manager which provides the ability to store supplemental information about the process step.

7.8 Process Modeller

Process modellers are used in the process domain to build a representation (process model) of selected processes corresponding to a problem domain for the purpose of understanding and communication among stakeholders. These modellers supports instantiation of a process meta-model (by means of an associated process language and diagramming notation) as well as a free representation of processes (by means of a process language and a diagramming notation). The level of details expressed in a process model depends on the stakeholders needs with regard to the process (for execution, more detail is necessary, for human consumption less detail is sufficient). Process modellers aid in obfuscating trivial details and presenting the essential details of the process (activities and flows). This enables the users of the process models to get a holistic understanding of the steps involved in achieving the desired outcomes. In essence, the process models provide a common ground for the role players of the process to understand and work cooperatively.
A process model is developed by stringing together a series of process steps in a specific sequence with the objective of delivering quality outcomes. Each step in the process is interdependent on another step in the process (output from one step becomes the input for another step, one steps needs to be completed for the other step to be performed and so on). As a result, many of these steps overlap on different factors and it is necessary to develop an appropriate view of the process by taking these concerns into consideration. Process modellers express the process by means of a process language. The extent of the success of modelling depends on the ability of the process language to capture the essence of the process and its process steps. There are five main characteristics of a process that a modeller captures as part of the model: a) Task/Activity/Process Step, b) semantics or meaning of the Task/Activity/Process Step, c) Sequence or flow of process steps/tasks/activities in relation to each other, d) information that is
necessary for performing the step, e) Specification of the work products that the step produces and f) inputs and constraints on the process step.

Accordingly, the modeller, as shown in Figure 7.4, is a decomposition of: a) Activity modeller for expressing the process steps to be performed by a role player, b) Information modeller for expressing the supplementary information pertaining to each process step, c) Resource modeller for expressing the resources and capabilities necessary for performing the process step, d) Work product modeller for expressing the outcomes and outputs that emanate from the performance of each process step, e) Input modeller for expressing the inputs necessary for performing the process step, f) Constraint modeller for expressing the constraints that affect the performance of a process step and g) Sequence modeller for expressing the order in which the process execution should occur. The different modellers use the syntax and notations provided by the underlying process language to express the different process elements. In cases where the underlying process language does not have appropriate constructs corresponding to the process element then those process elements are not captured as part of the process model.

7.9 Process Modeller Run-Time

As discussed in section 7.8, the process modeller aids in capturing different levels of detail of a process and uses different process languages and associated notations for this purpose. It utilizes the just-in-time services provided by the run-time for supporting data handling, event handling, dynamically plug and playing modellers and for service invocation. The runtime is designed to handle discovery of socket interfaces, connection to sockets, information exchange with sockets, event listening and dispatching to sockets and disconnection from sockets. It provides a gateway to the runtime environment which contains active components that can be interacted with during the execution of the platform. Once a modeller is plugged into the framework, the run-time serves as an abstraction layer that extends the capabilities of the framework by considering these modellers as constituent elements of the framework.

As a result, the modeller component is treated as par with a normal component that is part of the component framework. To facilitate this, the run-time utilizes the interface services offered by the interface manager to dynamically attach to and detach from the modeller socket. It utilizes the capabilities of the service manager to consume services offered by the platform and to provide services to the platform. It uses the capabilities of the data manager to exchange information pertaining to the process models that are manipulated by the modeller with the modelling platform. It utilizes the eventing services provided by the event manager to listen and dispatch the events that are triggered by the
modeller as well as the modelling framework. This decomposition of the run-time is presented in figure 7.5.

7.10 Discussions

The experimental system addresses its primary goal of enabling plug and play of process modellers by:

a) Encapsulating process modelling functionality by encoding them into the modeller component.
b) Changing the elements of the component framework while retaining the plug and socket interface mechanism.
c) Supporting different process languages and modelling notations by means of appropriate modellers.
d) Encoding the plug and play interface mechanism as socket and compatible plugs.
e) Encapsulating the plug capability as part of the modeller run-time component.
f) Encapsulating the socket capability as part of the component framework run-time component.
g) Dispatching the events to the appropriate handler by using the plug & socket interface.
h) Providing the services to the appropriate consumer by using the plug & socket interface.
i) Managing the model information by using the exchange services of the plug & socket interface.

The three critical elements, events, services and models that are required for proper functioning of the process modelling platform have been made accessible through the plug and socket interface.
However, the underlying component framework, as discussed in section 7.6, has been modified to address the needs of the process domain. Hence, it can be observed that “plug and play architectures are domain independent”. Accordingly, the hypothesis “If components are plug and playable, then irrespective of the domain, extensible systems become possible” was verified. Further, the underlying principles and constraints that are discussed in the subsequent sections enable the adoption of the plug and play based extensible architecture in process domain.

7.10.1 Principle of compatible support elements

According to Maier [51], the architecture of the supporting element should fit into the system that it supports. In the case of the process modelling platform, the plug interface of the modeller and the socket interface of the component framework are designed such that only compatible modellers can utilize this interface.

7.10.2 Principle of grouping and separation

According to Maier [51], while partitioning a system, it is necessary to group those elements that are strongly related to each other and separate elements that are unrelated. In the case of the modelling platform, the capabilities: a) Model rendering, b) Model storage and retrieval, c) Model transformation, d) Model traversal, e) Model assembly, f) Model extraction, and g) Model instantiation are common across all process models and hence the components that provide these functionalities are grouped together and co-exist as part of the component framework. However, the different modellers support individual process languages and diagramming notations and hence they are separated as pluggable components.

7.10.3 Principle of separation of concerns

According to Maier [51], while partitioning a system, it is necessary to leave the specialities to the specialist. Accordingly, in the modelling platform, while the generic capabilities like rendering, storage & retrieval, transformation, traversal and assembly are handled by the component framework, the core tasks pertaining to process modelling by adopting a process language is handled by the specially designed process modellers that are plugged into the component framework.

7.10.4 Principle of minimal communication

According to Maier [51], while choosing the sub-systems, it is necessary to define them in such a way that they are independent to each other and there is minimal communication between them.
Accordingly, the different modellers are independent to each other and they do not communicate with each other by any means. All their communication happens with the component framework and the constituent elements of the component framework.

7.10.5 Principle of accommodating change

According to Maier [51], in large systems, evolution is a process of ingress and egress. Accordingly, evolution in the case of the process modelling platform is made possible by the ability to plug a modeller; to plug an upgraded or new modeller; and to unplug a modeller that has outlived its expectations.

7.10.6 Principle of implementing change

According to Maier [51], when implementing a change, it is necessary to keep some of the elements constant so that people can still relate to it. Accordingly, after the process is modelled and there is a need to change the information associated to the process steps, the modeller allows the information meta-model associated to a process step to be changed and accommodates this change by smart transformation of available information.

7.11 The Plug and Play Architectural Abstractions

This section introduces and elaborates the plug and play architectural abstractions of the process modelling platform. While many of these abstractions were discussed in earlier experiments, their capabilities have improved at the end of this experiment, and are expressed as such. These abstractions are considered, based on the premise that “Software Plug and Play is a collection of standards, encoded in the run-time and embedded in the component schema, in order to support functional and structural extensibility of the process modelling platform”. These abstractions are:

7.11.1 Plug

A plug comprises of a collection of software resources that enables the plugged component to be executed on an underlying computing system. It serves as the gateway for the events, data and services that the plugged component manipulate as part of its computation and allows two way communication between the component framework and the plugged components. The plug encodes the interface mechanism for connecting to a compatible socket thereby enabling the plugged component to function as an external component of a component framework.
7.11.2 Socket

A socket is a handle that allows the plugged components to utilize the data, events and services that are exposed by the component framework. It facilitates two way communication of events, data and service invocation between the plug and the component framework. As an interface, the socket performs a type checking for compatible plugs to validate the connection and disconnects those that fail the validation.

7.11.3 Component Framework

A component framework defines specific interaction and composition standards that should be adopted by the constituent components and provides the set of elements required for supporting the execution of these components. It implements a component schema which describes the characteristics of the constituent components in terms of the services, service operations, events, and other metadata that it should support. It handles events raised by all of its constituent elements (including the plugged components) as well as dispatches events for processing by these constituent elements. It consumes services provided by all its constituent elements as well as provides services required by them.

7.11.4 Plugged Component

A plugged component extends and adds functionality to the component framework thereby enabling the evolution of the underlying system. It utilizes the run-time services provided by the plug to attach itself to the component framework. It processes various events that it receives as well as dispatches some of these events to the component framework. It consumes services offered by the component framework as well as provides services to it.

7.11.5 Event

An event is an encapsulated set of software actions that are triggered during component execution, along with a set of data that capture the computational state under which the event was triggered. Such events are triggered when a set of pre-defined execution conditions are met. They can be triggered inside the component framework and dispatched to the plugged component for processing or triggered inside the plugged component and dispatched to the framework for processing.
7.11.6 Service

A service is an encapsulated set of software functionalities offered by a component, along with a set of policies of usage that declare the information necessary for the successful execution of the service, that can be invoked by other components for different purposes.

7.11.7 The Plug and Play Component Model

The Plug and play component model is put together by considering the plug and play architectural abstractions that were discussed earlier.

![Diagram of Plug and Play Component Model]

**FIGURE 7.6: DECOMPOSITION OF PLUG AND PLAY COMPONENT MODEL**

Accordingly, the constituent elements of The Plug and play component model are: a) The component framework which provides the composition semantics, the logical structure for facilitating components composition, underlying structure in terms of interfaces, resources and protocols for the composition of components playing specific roles, b) The component framework run-time which serves as a socket and provides the necessary functionality for supporting composition of components, communication between components, manipulation of components, management of components performance, invocation of functions and services supported by the components and transformation of data from one form to another, c) The Repository which hosts the different work products emanating from the process modelling
platform, and d) The components manager which manages the plugged components and their respective run-times. Figure 7.6 provides a decomposition of this model. While the component framework hosts the functionalities and processes that are utilized by the end-users, its run-time provides functionality and resources needed by it, along with the ability to serve as a socket. It provides basic services necessary for various plugged components to work together, for raising and handling events, for consuming various services that are supported by the plugged component and for managing data flow between the plug and socket.

### 7.12 Summary

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled Process Modelling Platform. The approach to adopt a plug and play based extensible architecture in the process domain was illustrated by utilizing a set of pluggable components that are attached and detached to the platform on demand. This has been exemplified by the modifications to the underlying component framework, as discussed in section 7.6, to address the needs of the process domain. Accordingly, the hypothesis “If components are plug and playable, then irrespective of the domain, extensible systems become possible” is verified. Understanding the key architectural principles of this experiment helped identify the underlying plug and play architectural abstractions. This was culled out and expressed in the form of an abstract Plug and Play component model.