INTRODUCTION
1.1. Introduction

This chapter provides an introduction to our research work. The fundamental concepts underlying our work have been discussed here. Section 1.2 elaborates the Model Driven Architecture (MDA) approach to software development. Section 1.3 discusses the Cloud computing model in detail. Section 1.4 discusses 'interoperability' which is one of the primary goals of MDA, and also one of the key challenges to Cloud computing. This section also highlights the technology and concepts such as Web Services and Service Oriented Architecture (SOA) which play a significant role in achieving this goal. Section 1.5 focuses on the genesis of the problem and section 1.6 defines the problem by specifying the problem statement. Section 1.7 mentions the objectives of the research work. Section 1.8 describes the approach followed by the authors. Section 1.9 lists the contributions made by the authors consequent to the work carried out by them. Section 1.10 presents the layout of the chapters in this thesis.

1.2. Model Driven Architecture (MDA)

MDA®, an initiative by OMG®, was initially drafted in late 2000 and adopted formally in 2001. It is an open, vendor-neutral approach to the challenge of business and technology change [1]. It separates the business and application logic from the technology-specific code that implements this logic. The business and technology aspects of the application being independent of each other, each can evolve at its own pace. Thus, MDA shifts the focus of software development from the solution domain to problem domain and bridges
the semantic gap that exists between domain-specific concepts and the programming technologies used to implement them. It helps to enhance the rigor, productivity and manageability of software development process. The primary goals of MDA are interoperability, reusability and portability through architectural separation of concerns [2].

The software development process, in MDA, is driven by the activity of modeling the software system. The requirements of the stakeholders with respect to the application are captured and represented as models. These models are the prime artifacts in this model-driven approach to software development and are used for understanding, design, construction, deployment, operation, maintenance and modification of a system [1]. During the development process these formal, domain-specific models are transformed into models at next lower level of abstraction and eventually into implementation code using automated transformation tools. This helps to greatly reduce (a) the time and efforts required to build the system, (b) the errors generated in the development process, and (c) the efforts required to make changes in the system. Thus, MDA aims at using automated tools for building system independent models and transforming them into efficient implementations. The three cornerstone principles that govern MDA are (a) direct representation of the ideas and concepts in the business domain as well as the application domain as models, (b) increased reliance on computer-based automation for analysis, design and implementation of models, and (c) use of industry based standards to facilitate the development process. Domain-specific languages may be used to facilitate modeling within a particular problem domain [3][4].
1.2.1. MDA – Models and Views

As such, MDA specifies various models at different levels of abstraction to represent a system [1][5][6][7]. The models considered to be at its core, as depicted in Figure 1.1, are:

1.2.1.1. Computation Independent Model (CIM)

The Computation Independent Model or the domain model specifies the system at highest level of abstraction. It is intended to bridge the gap between domain experts and system experts. It represents the Computation Independent View (CIV) of the system which focuses on the requirements of the system.

1.2.1.2. Platform Independent Model (PIM)

The Platform Independent Model represents the system at the next lower level of abstraction. It specifies the functionality of the system independent of the platform to be used for its implementation. It represents the Platform Independent View (PIV) of the system which focuses on the operation of the system while hiding the details necessary for a particular platform.

1.2.1.3. Platform Specific Model (PSM)

The Platform Specific Model which is at the next lower level of abstraction describes the system with respect to the specific platform on which it would finally be implemented. It represents the Platform Specific View (PSV) of the system. The PSV combines the PIV with an additional focus on the detail of the use of a specific platform by a system. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.
1.2.1.4. Implementation Model

The Implementation Model represents the system at the lowest level of abstraction. It specifies the implementation details of the system and includes all the information required to construct a system and put it into operation. This model represents the generated code and auxiliary files that are ready for compilation, linking with legacy or other code, and deployment.

![Diagram of Abstraction Levels in MDA](image)

Figure 1.1 Abstraction Levels in MDA

The transformation of models from one level to another is performed using (semi)automated transformation tools defined for the purpose. A single PIM may be transformed into one or more PSMs that are targeted on different platforms. A platform, in this context, is defined as a set of subsystems/technologies that provide a coherent set...
of functionalities through interfaces and specified usage patterns. Any subsystem that depends on the platform can use these functionalities provided by the platform without concern for the details of how they are implemented [2].

1.2.2. OMG Modeling Standards and MDA

The Model Driven Architecture systematically builds upon several other OMG standards such as the Unified Modeling Language™ (UML), the Meta Object Facility (MOF), XML Metadata Interchange (XMI), Software Process Engineering Metamodel (SPEM), Common Warehouse Metamodel (CWM) and Business Process Modeling Notation (BPMN). MDA is different from other OMG standards (e.g. UML, CORBA) and non-OMG standards (e.g. Java and XML) in the sense that whereas these standards are specific in nature, MDA is a collection of specific standards and a way of thinking guided by a set of principles [8]. MDA leverages the widely accepted approaches such as design patterns, component-based development, and n-tier architecture and is acknowledged by several major system vendors such as IBM/Rational, Microsoft and Sun.

1.2.2.1. Meta Object Facility (MOF)

MOF, a core standard of OMG, is a platform-independent metadata management framework and forms a concrete foundation for MDA. The MOF-based structural, behavioral and data models of an application may be defined in a UML or a non-UML modeling language. These models can be exported to or imported from other applications, transported across a network, stored in a repository and then retrieved, rendered into different formats, transformed, and can be used to generate application code. MOF facilitates definition of different languages for modeling different aspects of systems and for integrating models expressed in different languages [9]. The technique
for defining the abstract syntax of models and the inter-relationships between model elements is referred to as meta-modeling [10]. MOF provides the five basic concepts of object-oriented programming for use in defining metamodels which include – classes, associations, data types, packages and constraints [11].

Based on MOF-enabled transformations, the MDA aims at unifying every step of the software development from PIM through PSM to generated code and deployable application. The PIM remains stable as technology evolves. The PSM can be extended to newer technologies by including implementation-specific details. This helps to maximize the Returns on Investment made for the software. In addition, portability and interoperability are incorporated into the architecture of the software system [12].

1.2.2.2. Unified Modeling Language (UML)

The Unified Modeling Language™ (UML) along with the Meta Object Facility (MOF) provides a key foundation for OMG's Model-Driven Architecture. It helps in unifying every step of development and integration from business modeling, through architectural and application modeling, to development, deployment, maintenance, and evolution [13]. It is the most well known and widely used graphical language for visualizing, specifying, constructing and documenting the artifacts of a software intensive system. With its rich set of diagrams, UML can graphically depict not only the structure, behavior and architecture, but also the business process and data structure of the system under discussion, from a specific viewpoint and at a specific level of abstraction. UML is also MOF-based [14][15][16]. A wide range of commercial as well as proprietary tools are available for implementing UML models. A few examples of UML tools are Rational Rose (IBM Software), Together ControlCenter (Borland), Poseidon (Gentleware), Objecteering (Objecteering Software, subsidiary of SOFTEAM), Enterprise Architect (Sparx Systems) and Eclipse UML2.0 Tools (Eclipse).
A profile, in UML, is an extension mechanism defined by a set of stereotypes, a set of tagged values and a set of related constraints [14][17]. Profiles are used to adapt UML to a particular platform like J2EE, .NET etc. or to a particular domain such as medical, financial, aerospace etc., by capturing their vocabulary and applying it to model elements making them more meaningful. An example of UML profile is EDOC profile. But, MDA certainly does not restrict itself to UML for modeling.

1.2.2.3. XML Metadata Interchange (XMI)

The XML Metadata Interchange (XMI), an OMG standard, is a model-driven framework for defining, interchanging, manipulating and integrating XML data and objects. XMI provides a mapping from OMG’s Meta Object Facility (MOF) to XML. It defines rules for generating an XML schema or DTD (Document Type Definition) from a valid XMI-transmissible MOF-based metamodel [18]. The schema or the DTD then serves as a format for encoding instances of the model. XMI is independent of any middleware technology and can be used for any metadata whose metamodels can be expressed in MOF. Besides being used as an interchange format for metadata, XMI allows the interchange of models in a serialized form. MDA views XMI as a significant information conveyance technology [19]. The XML serves as a means to serialize and de-serialize data among disparate systems, especially when the data is to be stored and transported across the Internet [20].

1.2.2.4. Common Warehouse Model (CWM)

The Common Warehouse Metamodel (CWM) is an MOF-based OMG standard. It provides a framework for representing metadata about data sources, data targets, transformations and analysis, and processes and operations that create and manage warehouse data. CWM metamodels allow users to trace lineage of data by providing
information that describes where the data came from and when and how the data was created. CWM primarily enables easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments. XMI is used to interchange data warehouse metadata based on the CWM metamodel. Instances of the metamodel are exchanged via XMI documents [21].

The CWM is a specification for modeling metadata for relational, non-relational, multidimensional systems and most other objects found in data warehousing environment. CWM defines the abstract syntax and semantics of modeling constructs in the metamodel. For example, the relational data metamodel of CWM defines familiar classes of relational modeling elements, such as table and column and the associations between them, used to specify data models [22].

1.2.3. Model Transformation

A model transformation is defined as a process of automatic generation of target model from a source model, according to some specific transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language [23]. Sendall et al in [24] defined model transformation as “a mapping of a set of models onto another set of models or onto themselves, where a mapping defines correspondences between elements in the source and target models”.

Models and model-based transformation are the essential parts of effective automated software development [25]. The key to the success of MDA lies in automating
transformations between models. Automated transformation tools should be able to perform model-to-model and model-to-code transformations. Figure 1.2 depicts the transformation of a higher-level, platform-independent business model (source model) into lower-level platform-specific models (target models). The transformation tool executes the transformation definition specified for the purpose.

![Figure 1.2 PIM to PSM Transformation](image)

A model describing a system must therefore be written in a well-defined language with a well-defined form (syntax) and meaning (semantics) so as to make it suitable for automated interpretation by computer [23]. Automation would help to greatly improve the productivity of developers and the quality of the models. With several interrelated models describing the application, significant efforts need to be made to ensure their overall consistency and synchronization. Besides checking inconsistencies and synchronizing models at same or different levels, automation will also help to refine models, re-factor models, reverse engineer models and generate new views from existing models [10].

### 1.2.4. Model Transformation and Meta Models

As discussed in section 1.2.2.1, Meta Object Facility (MOF) is the language definition mechanism of MDA [22][23]. It defines an abstract syntax of modeling constructs for specifying, constructing, and managing technology-neutral metamodels; the technique is referred to as metamodeling. The models in MDA must conform to their respective
metamodels. The OMG's Meta Object Facility (MOF) standard facilitates this conformance.

A four-layered MOF architecture [12], starting from bottom to the top, comprises of – the data/information layer (M0), the model layer (M1), the metamodel layer (M2), and the meta-metamodel layer (M3). Each ith layer in this architecture consists of a set of constructs which are used to define the elements of (i-1)th layer. While a model at M1 layer is an abstraction of a real world phenomenon described at M0 layer, a metamodel at M2 layer is an abstraction of the model at M1 layer. The M3 layer essentially models the technology neutral metamodels at M2 layer. The MOF M3 layer is self-describing i.e. the M3 elements are instances of M3 elements. The MOF is, therefore, a closed meta-modeling architecture.

---

**Figure 1.3 Model Transformation in MDA**
Figures 1.3 depicts the model transformation in MDA based on MOF standard. A transformation tool transforms a source model into a target model by executing a transformation definition specified for the purpose. The source and target models must conform to their respective metamodels. The transformation definition is described using a transformation language (metamodel) which is chosen in such a way that it maps the constructs of source metamodel into constructs of target metamodel. In a PIM to PSM transformation, the PIM is the source model and the PSM is the target model. The PIM and PSM are specified at the M1 level of the MOF architecture, and must conform to their respective metamodels at M2 level.

Mapping functions alone may not always be sufficient to transform a source model completely and additional inputs may be required to perform the mapping. These additional inputs are provided as *marks*. Marks are lightweight, non-intrusive extensions to models that capture information required for model transformation without polluting those models [26]. Marks are annotations that are used to guide the transformation of a PIM to a PSM and may be provided by types or roles in model, stereotypes in a UML profile, and elements in a metamodel or an MOF model. A mark represents a concept in
the PSM. As depicted in Figure 1.4, a particular platform on which the PIM is to be targeted is chosen. A mapping for this platform is either available or prepared. This mapping includes a set of marks. These platform-specific marks are applied to the elements of the PIM, to indicate how these elements are to be transformed. The marked PIM and the mapping constitute the input to the transformation process resulting in a PSM and a record of transformation as output [2][26][27].

Koehler et al [28] have presented a model-driven transformation method to map the business view models (representing PIM) into IT architectural models (representing PSM), and vice versa. The approach is based on generating process graphs and flow graphs.

Model transformations are mostly developed with modeling tools using proprietary languages. As a result, the transformations are far from being robust and reusable. In the light of MDA, Bézivin et al. [29] have proposed ‘Reflective Model Driven Engineering’ whereby the transformations should be developed along a cycle ranging from platform-independent transformations down to platform-specific transformations.

1.2.5.Model Transformation Classification Approaches

A variety of approaches for classification of model transformation from different perspectives have been proposed in literature.

Mens and Gorp in [30] proposed a multi-dimensional classification approach that allows grouping or comparing model transformations based on the criteria of interest. Their taxonomy is targeted more towards tools, techniques and formalisms supporting the model transformation activity.

Czarnecki and Helsen in [31] described a classification of various transformation approaches using a feature model which is a combination of feature diagrams and
Introduction

additional information. They broadly categorized transformations into model-to-model and model-to-code transformation approaches. The former included (a) visitor-based approach, and (b) template-based approach; while the later included (a) direct-manipulation approach, (b) relational approach, (c) graph-transformation based approach, (d) structure-driven approach, and (e) hybrid approach.

Zhao and Zhang in [32] described the model-transformation approaches based on four dimensions – underlying formalism, application, domain language support and modeling layer. The classification was based on a multidimensional scheme.

Prakash et al in [33] proposed a framework which considers the model transformation approaches along three different views or dimensions. Each view allowed capturing a particular relevant aspect of each of the transformation approach. The three views or dimensions were: technique view, language support view and generic view.

Based on the architecture of transformation tools, Sendall and Kozaczynski in [10] classified transformations into (a) the direct model manipulation approach, (b) the intermediate representation approach, and (c) the transformation language support approach.

1.2.6. Characteristics of Model Transformation

A number of features are desirable in a transformation process [23]:

1. Tunability: It refers to the ability of a general rule defined in a transformation definition to be tuned to the specific needs of the user.

2. Traceability: It refers to the ability to trace an element in the target model back to the element(s) in the source model from which it is generated.
3. Incremental consistency: This means that when target-specific information has been added to the target model and it is regenerated, the extra information persists.

4. Bi-directionality: This means that a transformation can be applied not only from source to target, but also back from target to source.

Sendall and Kozaczynski in [10] recommended a set of characteristics that a model transformation language supporting model driven software development must possess. The transformation language must be – executable, fully expressive, and implementable in an efficient way, easy-to-understand, precise, concise, unambiguous, complete, graphical, declarative and easy-to-modify.

In order to make the quality of model transformation measurable, van Amstel et al in [34] defined a number of attributes that must be possessed by model transformations. These attributes include understandability, modifiability, reusability, reuse, modularity, completeness, conciseness and consistency.

A complete MDA specification consists of a definitive platform-independent base model, plus one or more platform-specific models (PSM) and sets of interface definitions, each describing how the base model is implemented on a different middleware platform [35].

MDA development focuses on the functionality and behavior of the distributed application or system, unaffected by the technology to be used for its implementation. Thus, the PIM depicting the structure, behavior and functionality is modeled only once. And then, the automated transformation definitions enable the transformation of the PIM to one or more PSMs [36][37].

1.2.7. Key Benefits of MDA

The prime benefits of MDA can be attributed to its underlying key principles [38][39] which may be summarized as: a) use of formal models, expressed in a well-defined
notation, to describe the system, b) defining models at different levels of abstraction and specifying transformations between them, c) use of metamodels to describe models and automated model transformations, and d) adoption of industry standards to facilitate software development process.

The primary benefits of software development based on MDA approach include:

1. **Improved Quality**

The models and model transformations are defined in a standard language based on OMG standards thereby improving the quality of the system design and implementation, particularly as the solution evolves through maintenance and upgrade.

2. **Reusability**

MDA encourages reuse of known patterns of behavior, building on existing architectural designs, and leveraging expertise more effectively. Reuse also improves software development productivity and quality.

3. **Enhanced Productivity**

The models are defined at different levels of abstraction to represent various aspects of the system. The developer needs to focus only on the business logic and find an effective and efficient solution to the business problem at hand, rather than going into the details and specifics of target platforms. The platform-specific implementation details are included in the transformation definition. The repetitive, time-consuming tasks are automated thereby enhancing the productivity of the developers and of the systems.

4. **Interoperability**

A technology-independent PIM specifying the function and behavior of the system can be transformed into several PSMs targeted on different platforms. The cross-platform
interoperability of the product is realized by defining tools that not only generate PSMs, but also the bridges between them. The established modeling standards of OMG such as UML, XMI, MOF and CWM enable the MDA to provide an open, vendor-neutral approach to system interoperability.

5. Portability

The focus, in MDA approach, is on developing platform-independent models of the system under consideration. The portability is achieved by transforming PIM to various PSMs targeted for specific platforms. The extent to which portability may be achieved depends on the available automated transformation tools. In the absence of an available transformation tool, a tool supporting a plug-in user-defined transformation specification may be used.

6. Rapid Inclusion of Evolving Technologies and Improved ROI

New and better technologies are evolving in the software industry at a rapid rate and the enterprises are required to keep pace with them for reasons such as customer demand, changing requirements, obsolescence etc. This makes most of the earlier investments worthless. In MDA, the models drive the software development activity. As the technology evolves, the PIM remains stable. The PSMs are extended to include newer technologies, thereby maximizing Return on Investment. The platform independence thus makes it easier to deal with technology volatility.

7. Reduced time to market

In MDA, the models are the prime artifacts representing various stages of software development. The model-to-model and model-to-code transformations are performed using (semi)automated transformation tools. By virtue of automation, the software products are developed faster, significantly reducing the time to market the final product.
8. **Reduced cost**

The reduction in the cost of software development is attributed to the use of formal models, automation of transformation and other process, pattern-based development and reuse of the artifacts. The generation of code from the models is automated. Test scripts may be automatically generated from the models during the testing phase. Also, as the requirements of the user change over a period of time, the changes are incorporated in the models from which the implementations are automatically generated. This greatly reduces the development as well as maintenance cost of the system. Since the models are documents of the system in themselves, the organization need not invest in documentation separately.

9. **Maintenance and Documentation**

The traditional software development process documents the system mainly in the form of text and diagrams which are created during the early stages of the development process. Once the implementation begins, these documents start losing their importance and are finally abandoned. As the system evolves, the changes are incorporated at the code-level only. In due course of time, the document and implementation become two unrelated entities making it extremely difficult to manage and maintain the system. In model-driven approach, the PIMs which are at higher level of abstraction serve the function of high-level documentation and remain consistent with the actual code. Any modification in the system requires the PIM to be modified from which the PSM and code are regenerated.

10. **Improved communication**

As models are used to describe different aspects of the system, this improves communication among the stakeholders as well as the models themselves; and at the
same time helps to understand the system better. The requirements of the users are better understood. This may ultimately lead to developing better systems that would fulfill the requirements of the user.

1.2.8. Challenges in Adoption of MDA

In spite of being a promising technique for software application development, MDA has not been able to penetrate deep in practice. The most common objections to the adoption of MDA include its relatively more complex and academic nature; the belief or, perhaps, myth that MDA is just about code generation and is quite rigid in its approach [40]. The key impediments that are restricting a more extensive realization of this vision have been identified and categorized into technical hurdles, cultural hurdles and economic hurdles in [41]. The technical hurdles are (i) poor usability of MDA tools which, although diverse and powerful, are invariably difficult to learn and to use, and (ii) lack of sound theoretical basis for model-driven development. The cultural hurdles are (i) insufficient awareness of the practitioners with regard to the potentials and capabilities of MDA, and (ii) conservative mindset of the practitioners and resistance to technological change. The model-driven methodology professes long-term investment in software development methods and tools which is in contrast to the prevailing business environment focused on short-term Return on Investments. This poses an economic hurdle in the adoption of Model Driven Architecture.

1.3. Cloud Computing

Cloud computing, an evolving computing paradigm, is reshaping the world of Information and Communication Technology (ICT). The term cloud was first used in early ‘90s to refer to large ATM networks. Cloud computing began in earnest at the onset of this century with the advent of Amazon’s Web Services [42]. The other big
vendors who gradually joined the fray include Yahoo, Google, Microsoft, IBM, Sun, Intel, Oracle and Adobe. The first to give prominence to the term ‘Cloud computing’ was Google’s CEO Eric Schmidt, in late 2006 [43]. The term ‘cloud’ is a metaphor for the Internet which is graphically represented as a cloud. Thus a cloud, in the given context, refers to a complex, internet-based infrastructure of hardware and software components. The key attributes of Cloud computing are summarized in Table 1.1.

Table 1.1 Key Attributes of Cloud Computing

<table>
<thead>
<tr>
<th>S.No</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Infrastructure systems</td>
<td>It includes servers, storage, and networks that can scale as per user demand.</td>
</tr>
<tr>
<td>2.</td>
<td>Application systems</td>
<td>It provides web-based user interface, Web services APIs, and a rich variety of configurations.</td>
</tr>
<tr>
<td>3.</td>
<td>Application development and deployment software</td>
<td>It supports the development, integration and deployment of cloud application software.</td>
</tr>
<tr>
<td>4.</td>
<td>System and application management software</td>
<td>It supports rapid self service provision and configuration, and usage monitoring.</td>
</tr>
<tr>
<td>5.</td>
<td>IP networks</td>
<td>They connect end users to the cloud and the infrastructure components.</td>
</tr>
</tbody>
</table>

In the ambit of Cloud computing, a variety of computing resources such as processors, storage, software applications, databases, development environments etc are made available to the clients as services, over a high speed network. These resources in the cloud may be accessed as per demand across a simple interface such as a browser running on devices ranging from desktop, laptop, iPhone to even a PDA. The usage is metered and the client may choose to either pay for every single use of the service or subscribe to it for a definite period [44][45]. The resources are pooled and shared among the clients, thereby ensuring their optimal utilization. These resources are dynamically
scalable and are acquired from the pool or released to the pool with great elasticity, in order to meet the varying demands of the clients. Table 1.2 summarizes the vital attributes of cloud services.

Table 1.2 Key Attributes of Cloud Services

<table>
<thead>
<tr>
<th>S.No</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Offsite. Third party provider</td>
<td>In the cloud execution, it is assumed that third-party provides services. There is also a possibility of in-house cloud service delivery.</td>
</tr>
<tr>
<td>2.</td>
<td>Accessed via the Internet</td>
<td>Services are accessed via standard-based, universal network access. It can also include security and quality-of-service options.</td>
</tr>
<tr>
<td>3.</td>
<td>Minimal or no IT skill required</td>
<td>There is a simplified specification of requirements.</td>
</tr>
<tr>
<td>4.</td>
<td>Provisioning</td>
<td>It includes self-service requesting, near real-time deployment, and dynamic and fine-grained scaling.</td>
</tr>
<tr>
<td>5.</td>
<td>Pricing</td>
<td>Pricing is based on usage-based capability and it is fine-grained.</td>
</tr>
<tr>
<td>6.</td>
<td>User Interface</td>
<td>User interface include browsers for a variety of devices and with rich capabilities.</td>
</tr>
<tr>
<td>7.</td>
<td>System Interface</td>
<td>System interfaces are based on Web services APIs providing a standard framework for accessing and integrating cloud services.</td>
</tr>
<tr>
<td>8.</td>
<td>Shared resources</td>
<td>Resources are shared among cloud services users; however via configuration options with the service, there is the ability to customize.</td>
</tr>
</tbody>
</table>

1.3.1. Cloud Service Models

Cloud computing offers various computational resources as cloud services. Cloud services are analogous to utility services, and offer computing resources as services to the clients. Based on the type of resource offered as a service, the cloud service models are broadly categorized as [46] – Cloud Software-as-a-service (SaaS), Cloud Platform-as-a-Service (PaaS) and Cloud Infrastructure-as-a-Service (IaaS).
1.3.1.1. Cloud Software-as-a-Service (Cloud Saas)

Cloud Software-as-a-Service (SaaS) provides the consumer with the capability to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

A cloud SaaS is a multi-tenant platform that uses common resources and a single instance of both – the object code of an application as well as the underlying database – to support multiple customers simultaneously. It is based on the Application Service Provider (ASP) model and heralds a new era in application software distribution. The key providers of this service are Salesforce.com, NetSuite, Oracle, IBM, Microsoft etc.

1.3.1.2. Cloud Platform-as-a-Service (Cloud Paas)

Cloud Platform-as-a-Service (PaaS) provides the consumer with the capability to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations. The examples of this service are Google’s App Engine (GAE) and Microsoft’s Windows Azure.

1.3.1.3. Cloud Infrastructure-as-a-Service (Cloud IaaS)

Cloud Infrastructure-as-a-Service (IaaS) provides the consumer with processing, storage, networks, and other fundamental computing resources where he is able to deploy and run
arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating system, deployed applications, and possibly limited control of select networking components (e.g., host firewalls). The examples of this service include Amazon EC2 (Elastic Cloud Computing) Service and S3 (Simple Storage Service).

In the absence of a standard taxonomy, other suggested categories include Hardware-as-a-Service (HaaS), Development, Database and Desktop as a Service (DaaS), Business-as-a-Service (BaaS), Framework-as-a-Service (FaaS), Organization-as-a-Service (OaaS) etc [44][45].

1.3.2. Cloud Deployment Models

A cloud may be deployed as [46][47]:

1.3.2.1. Private cloud

The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

1.3.2.2. Community cloud

The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

1.3.2.3. Public cloud

The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.
1.3.2.4. Hybrid cloud

The cloud infrastructure is a composition of two or more types of deployments of clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

1.3.3. Characteristics of Cloud Computing

The key attributes that characterize the Cloud computing paradigm include [46][48][49][50]:

1. Ubiquitous Access of Resources – The services are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

2. Resource Pooling and Sharing – The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

3. Dynamic Scalability of Resources – The services can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

4. On-Demand Availability of Resources – A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring service provider’s intervention.
5. *Metered Usage of Resources* – Cloud systems automatically control and optimize resource usage by leveraging a metering capability. Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.

1.3.4. **Key Challenges to Cloud Computing**

To fully realize Cloud computing as a public utility certain significant issues related to deployment, efficient operation and use of Cloud computing infrastructure need to be addressed. Some of these are [47][50][51]:

1. **Software/Hardware Architecture** – The emergence of Cloud computing services suggests changes in hardware and software architectures to support systems with high degree of parallelism and data-intensive computing. Besides the storage technologies must be able to provide reliable and high performance data storage.

2. **Data Management** – Data management is important as the data is stored off-site, on third-party resources creating enormous risks for data privacy. Besides, the data is replicated across large geographical distances where its durability and availability are paramount for cloud service providers. Novel data management approaches are therefore desired to address data limitations.

3. **Cloud Interoperability** – Cloud interoperability refers to the ability to use the same software artifacts with a variety of Cloud computing providers and platforms. Therefore new standards and interfaces are required to be built that would enable the applications to interact with each other.

4. **Service Level Agreement** – Services under any cloud are required to meet stringent Service Level Agreement (SLA) in a business environment. These agreements are signed between the service providers and the consumers/end users. The agreement
may include issues concerning service provisions, legal issues and in addition how much bandwidth, CPU and memory the consumer can use at any given time throughout the day.

5. **Security and Privacy** – As the enterprise data is stored off-site, with a third party, the privacy of data raises serious concern. Besides, security threats may also occur during resource provisioning and during distributed application execution. Cloud services should be able to integrate new data protection mechanisms to preserve and secure data integrity, data privacy, resource security and content copyrights.

### 1.3.5 Cloud Computing and its Enabling Technologies

Cloud computing is the IT foundation for cloud services and comprises of technologies that enable cloud services. It has evolved from a variety of technologies and concepts that include distributed computing, grid computing, virtualization, Service Oriented Architecture (SOA), Software-as-a-Service (SaaS), Web Service with its supporting standards such as Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI) etc.

#### 1.3.5.1 Grid Computing

The grid computing technology matured into a well established technology in the early 2000s. Grid computing enables efficient management of geographically distributed computational resources that support different hardware and software configurations. These resources distributed across multiple administrative domains have their own policies and goals [52]. The large computing and storage capacity offered by grid technology led to the development of another category of services that later came to be identified as cloud services [43]. Grid computing applies the resources of numerous computers in a network to simultaneously work on a single problem in order to address a
scientific or a technical problem. It thus necessitates the use of software that can divide and then send out pieces of programs to thousands of computers.

1.3.5.2. Virtualization

Virtualization technology abstracts the coupling between the hardware and the operating system. The logical resources are abstracted away from the underlying physical resources in order to improve agility, flexibility, reduce costs and thus enhance business value [45]. It allows for dynamic allocation of resources as the resources can be acquired from or released to the pool of shared resources in response to the varying demands of the user. A sudden increase in demand of resources is met by diverting existing resources from a low-priority to a high-priority application [53]; thus, giving an illusion of the existence of infinite resources to the cloud service consumers. Virtualization in a cloud may be of different types – server virtualization, storage virtualization and network virtualization. It is therefore extremely well suited to a dynamic cloud infrastructure, as it provides important advantages in sharing, manageability and isolation of resources.

1.3.5.3. Web Services

The Web service technology refers to a pure web-based, distributed technology that leverages the concept of a standardized communications framework to bridge the enormous disparity that exists between and within organizations [54]. The most important part of a Web service is its public interface that assigns the service an identity and enables its invocation. The method invocation is performed by creating an XML message that encapsulates method name and any required parameters. Similarly, the value(s) returned by the method are wrapped in another XML message and sent back to the client excluding any presentation markups. The client’s browser extracts the information from the received message, adds presentation details and renders it to the
client. The Business-to-Business (B2B) transactions are, therefore, the domain of web services.

1.3.5.4 Service Oriented Architecture

Service Oriented Architecture is an architectural paradigm and discipline that may be used to build infrastructures enabling those with needs (consumers) and those with capabilities (providers) to interact via services across disparate domains of technology and ownership. Services act as the core facilitator of electronic data interchanges, yet they require additional mechanisms in order to function. Several new trends in the computer industry rely upon SOA as the enabling foundation [55]. And Cloud computing is one among them. The cloud should not be looked at as a new architecture, but instead as another option of storing and running services within SOA [56]. As the software applications in the cloud may be implemented on varied platforms and technologies, an SOA-based cloud can naturally abstract and hide the vendor-specific and platform-specific aspects of the various software applications by leveraging the open Web services communications framework and establishing a predictable communications medium for all applications exposed via Web service. A quality 'Cloud' therefore requires the individual cloud services to conform to common design principles of SOA in order to fully realize the benefits of reusability, interoperability, federation, and others [57].

1.3.5.5 Software-as-a-Service (SaaS)

Software-as-a-Service (SaaS) is a software application based on multi-tenant model. It is owned and managed remotely by one or more service provider(s) and delivered to the customers over a network on a shared, pay-per-use or subscription basis. A SaaS application must be elastic and scalable in order to qualify as a true Cloud computing service [58].
Thus, rather than competing with related paradigms and technologies, Cloud computing need to converge with the past developments so as to produce unified and interoperable platforms for delivering IT services as the fifth utility (the other four being water, electricity, gas and telephone) to individuals, organizations and corporations [52][59].

1.3.6. Advantages of Cloud Computing

Cloud computing addresses the next evolutionary step of distributed computing with the goal to make a better use of distributed resources, putting them together in order to achieve higher throughput and be able to tackle large scale computation problems. It allows for the most cost effective development of scalable web portals on highly available and fail-safe infrastructures [45].

Cloud computing is especially beneficial to small- and medium- enterprises (SMEs) for attaining supercomputing capabilities without incurring any capital expenses on purchase of infrastructure, purchase of software licenses, signing of agreements or hiring trained professionals. Instead, a single application loaded into the computer allows the user to log into a web-based service executing on remote machines, which host all the programs required for the job. The key component in the Cloud computing system is the interface software which may be as simple as a web browser. A central server administers the system, monitoring traffic and client demands to ensure that everything runs smoothly.

Some benefits of Cloud computing are:

- Clients can access their data and applications from anywhere at any time.
- It reduces the need for advanced hardware on client side.
- Organizations have company wide access to the applications.
- It removes the need for physical space on the front end.
- Clients can achieve supercomputing powers from minimal resources.
• It eliminates cost and complexity of buying, configuring and managing the resources needed to build and deploy applications which in turn are delivered as a service.
• It purges the need to hire specialized man power for development of complex systems at the front end.
• The cost incurred on training the personnel as the technologies change, is also minimized.
• It promotes green computing.

1.4. Interoperability

As discussed in the previous section, one of the key challenges to Cloud computing is interoperability among the applications that are hosted within the same cloud or among different clouds. The issue becomes significant in the light of diverse platforms and IT infrastructures used for implementing individual applications in the clouds. Besides, as mentioned earlier, interoperability is one of the primary goals of Model Driven Architecture, in addition to portability and reusability.

The IEEE Glossary [60] defines interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged”. The applications must be able to access information transparently irrespective of their underlying operating systems, hardware and software technologies, and networking infrastructures.

Service Oriented Architecture (SOA) fosters intrinsic interoperability and Web Services framework, at present, provides the necessary technology to realize SOA. Thus, SOA along with Web Services framework helps to resolve the interoperability issue. The solution is based on the use of standard protocols and technologies like HTTP, XML and SOAP that are widely used in the Internet to address interoperability between the various
applications.

1.4.1. Web Services

A web service is defined as any service that is available over the Internet or an intranet, uses standardized XML messaging system and is self-describing, discoverable and not tied to any operating system or programming language [61]. Michael Rosen in [62] defined web service as “a software construct that exposes business functionality using Internet-related technologies.”

The web-services communications framework comprises of three XML-based standards – Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI). WSDL is an XML-based document that describes a web service. It specifies the location of the service and the functionalities exposed by the service. SOAP is an XML-compliant communication protocol that specifies the message format for exchanging information among the applications in a distributed environment. The service description registry and discovery is realized through UDDI. Thus, while WSDL specification provides a means to expose the details of how to interact with the service and its methods, SOAP provides a protocol for calling remote methods and passing complex data types [57].

The emergence of these standards as a means of communication between clients and service providers finally allow both parties to operate in a truly platform-independent manner. Since these standards are open, non-proprietary and freely available, they are readily endorsed by different vendors. The use of these standards provides neutral ‘language’ for heterogeneous environments to communicate with each other. This allows for easy interoperability allowing data and/or information to be accessed transparently by the software applications written in various languages, on different operating systems,
and running on myriad devices such as personal computers, laptops or a handheld device such as a smartphone or a PDA.

1.4.2. Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) represents an architectural style in which the automation logic of an enterprise application is decomposed into smaller, distinct units of logic called services. These services are then configured to provide business solutions. These units of logic in SOA must conform to the technology-independent principles of loose-coupling, autonomy, abstraction, reusability, discoverability, statelessness, adherence to a service contract and composability [54]. The services themselves may be distributed. Each service implements well-defined business functionality and has a well-defined interface. The interface isolates the service implementation details from the client. SOA is an implementation-agnostic paradigm that can be realized with any suitable technology platform. Currently, the Web Services platform with its supporting standards is the choice for realization of SOA. The services in SOA are, therefore, inherently interoperable.

1.5. Genesis of Problem

The field of Information and Communication Technology (ICT) is a dynamic field with rapid advancements. As is evident, the hardware and software technologies are constantly evolving. Keeping pace with the emerging technologies, the computing paradigms themselves have evolved from mainframe computing, through personal computing, network computing, distributed computing, grid computing to the most recent, the Cloud computing. Cloud computing architectures are a heterogeneous blend of technologies and platforms. The various software applications residing in the cloud do not exist in isolation. They must be able to communicate with each other and exchange
information transparently irrespective of the technologies used to implement them. Thus, interoperability among the cloud SaaS is a relevant and significant issue in Cloud computing.

With the technologies continually evolving, the IT industry is persistently faced with the challenges of technology obsolescence. In reference to Cloud computing, the legacy software applications in the cloud may become archaic despite their business functionalities remaining relatively constant. The changing technologies have more serious ramifications in B2B context. This incurs additional expenses on part of the service providers. Therefore, it becomes essential to promote a technology agnostic software development approach that could alleviate the undesirable effects of technology change; the one that could cope with multiple implementation technologies and extend the lifetime of the cloud software application.

In this perspective, Model-driven Architecture (MDA) becomes a preferred methodology for developing cloud software services. An MDA based development of cloud SaaS (application) will enable defining these services in a technology-independent manner and will play a significant role in improving the quality of cloud software services, making them more robust, flexible and agile [58]. Encapsulating business logic in a manner that is independent of the technical mechanisms will formally capture the essence of the applications; and will also make it possible to reuse them in a variety of contexts [63].

Our work presents an MDA-based model-driven approach to develop cloud software services and to show how interoperability among these services can be ensured.
1.6. Problem Definition

*To Ensure Interoperability of Services under SaaS in Cloud Computing Environment by Maintaining Platform Independence Through Separation of Concerns Using PIMs and PSMs.*

1.7. Objectives

1. To produce PIM of a cloud software services (Cloud SaaS) deployed in an SOA-based cloud.

2. To produce PSM of the Cloud SaaS from the PIM.

3. To define and implement the interface for exhibiting interoperability among the services.

1.8. Approach Followed

In an attempt to find a solution to the problem, the first and foremost step was to conduct a thorough study of the fundamental concepts of Model Driven Architecture (MDA). This included an in depth study of the various models such as CIM, PIM and PSM, the metamodels and the model transformation approaches available thereof.

Since, the objective of the research was to leverage the MDA approach for the development of Cloud SaaS, a comprehensive study of the Cloud computing paradigm along with its enabling technologies especially the Web services and SOA was carried out next.

Subsequently, a hypothetical cloud software application residing in the cloud (a Cloud SaaS) – an Online Hotel Reservation System (OHIRS) was identified. Based on MDA, a
CIM of the OHRS was specified using Use Case diagram and Activity Diagram in UML. Next, a PIM of OHRS application was specified using the Class diagram in UML. This PIM was created manually without using any automation tool. Given that a single PIM in MDA can be transformed into several PSMs targeted on different platforms, the transformation definitions for transforming the OHRS PIM to its Relational PSM and a Java PSM were specified next.

In view of the fact that a software application running in the cloud is offered to the clients as a service with a well-defined interface, an exhaustive study of the Web services communications framework comprising of WSDL, SOAP and UDDI was conducted next. As WSDL is used to define the service interface, the elements of WSDL were studied comprehensively.

The CIM of the OHRS application specified the functionalities to be offered by the cloud service to its clients through a well defined interface. The clients here can also be other services in the cloud. This CIM was therefore used to specify the PIM of the OHRS cloud service interface. Again, the PIM was constructed manually without the use of any automation tool. Next, this PIM was to be transformed into a PSM targeted on WSDL 2.0. An essential requirement in MDA is that the source model and target model must conform to their respective metamodels. Therefore, a metamodel for WSDL 2.0 was developed. A transformation definition was then specified to transform the PIM of OHRS Cloud SaaS to its WSDL PSM conforming to the WSDL metamodel defined earlier. The automated transformation of PIM to WSDL PSM was performed using a tool developed by the authors for the purpose.
Introduction

Study of concepts of MDA – Models, Metamodels and Model Transformation

Study of concepts of Cloud Computing and its enabling technologies and concepts such as SOA and Web Services

Identification of a Cloud SaaS - an Online Hotel Reservation System (OHRS)

Specification of Models for OHRS Cloud SaaS (Application)

PIM to Relational
PIM to EJB PSM

Implementation of PIM to Relational PSM transformation

Data Modeling

Definition of OHRS Cloud Service Interface to allow interoperability

Specifications of Models for OHRS Cloud Service Interface

Specification of WSDL Meta Model

Specification of Transformation Definition (PIM to WSDL PSM)

Implementation of Transformation Definition (PIM to WSDL PSM)

Service modeling through MDA ensuring interoperability

Figure 1.5 Work Flow Diagram of Research Work
The primary objective of our research is to ensure interoperability among the Cloud SaaS, based on a model-driven approach. The Web services communications framework facilitates the interaction among the services. Thus, defining a PIM for the cloud service interface and automatically transforming it into one or more corresponding PSMs targeted on available technologies, such as the Web Services framework at present, would ensure interoperability between the services despite changing technologies.

A Workflow diagram presented in Figure 1.5 pictorially represents the approach followed to accomplish the task.

1.9. Contributions

Our work is inspired by the necessity to leverage the MDA approach in the development of Cloud software services, in order to mitigate the undesirable affects of technology obsolescence which is one of the major challenges faced by the software industry. Since, the software services in the cloud do not exist in isolation and require to interact with each other in order to achieve a common business goal, 'interoperability' which is a significant issue is addressed in this work. The major contributions of our work presented through this thesis are:

1. Definitions of well-defined models of a cloud software application based on the Model Driven Architecture (MDA) approach.

Chapter 2 is inspired by the fact that constantly evolving hardware and software technologies pose a major challenge to the software industry in the form of technology obsolescence. A software development approach based on MDA, wherein models are the primary development artifacts, may help to mitigate the undesirable affects of technology change. Two research papers titled 'Cloud SaaS and Model Driven Architecture' and 'Modeling Cloud Software-as-a-Service: A Perspective' published in *Proceedings of the*
2. Specification of Transformation definitions to transform a PIM to its Relational PSM and EJB PSM

The success of MDA is attributed to the automation of model-to-model (M2M) and model-to-code (M2C) transformations. Also, in MDA, a single PIM may be transformed into one or more PSMs. Chapter 3 specifies the transformation definitions for transforming a PIM to its Relational PSM and Java PSM. A research paper titled ‘Cloud SaaS: Models and Transformation’ included in the Proceedings of First International Conference on Computer Science, Engineering and Information Technology (CCSEIT-2011) [65] published by Springer (LNCS) in Communication in Computer and Information Science (CCIS) Series, presents the PIM to Relational PSM transformation definition for a cloud SaaS taken as example. The transformation definition for transforming a PIM to EJB PSM is presented in ‘Enhancing Cloud SaaS Development with Model Driven Architecture’ has been published in November 2011 Issue of International Journal on Cloud Computing: Services and Architecture (IJCCSA) [66].

3. Approach to establish interoperability among the software applications in a cloud based on Service Oriented Architecture (SOA) and Web services communications framework.

One of the primary goals of MDA is ‘interoperability’ among the various software applications. Chapter 4 underlines the need to develop clouds based on SOA paradigm and realize interoperability among the cloud software applications through Web services communications framework. This is highlighted in two research papers titled ‘Modeling

4. Definitions of well-defined models of Cloud software service interface representing the functionalities provided by the service, based on MDA approach, enabling communication among services and hence ensuring interoperability.

MDA requires the specification of well-defined models for the development of software. The service interfaces that describe the functionalities offered by a service are also software components and must be modeled. Based on MDA approach, the models of service interface for a cloud SaaS taken as example are presented in Chapter 5. Two research papers titled ‘Modeling Cloud SaaS with SOA and MDA’ included in the Proceedings of International Conference on Advances in Computing and Communications (ACC-2011) [67] and ‘A Model-Driven Approach to Cloud SaaS Interoperability’ published in ‘International Journal of Computer Application’ September 2011 edition [57], underline this contribution.

5. A Meta model for Web Service Description Language (WSDL2.0)

The MDA approach emphasizes on the requirement for various models, specified for a software system, to conform to their respective metamodels. In this regard, the authors present a WSDL metamodel. The WSDL PSM of the cloud service must comply with this metamodel. In this thesis, the WSDL Metamodel is presented in Chapter 5. A

6. **Specification of a transformation definition to transform the PIM of cloud software service to its WSDL PSM.**

As mentioned, the key to success of MDA lies in the automation of various transformations that transform models from one formalism to another. A research paper titled 'A Model-Driven Approach to Cloud SaaS Interoperability' published in the September 2011 edition of *International Journal of Computer Application* [57] describes the transformation definition for transforming the PIM into a WSDL PSM representing the abstract description of the service based on Web service communications framework. Chapter 5 presents this transformation definition in this thesis.

**1.10. Layout of Thesis**

Figure 1.6 pictorially displays the layout of the thesis.

**Chapter 1** elaborates the fundamental concepts and technologies that constitute the research work. It presents a detailed description of the MDA approach, the Cloud computing paradigm and its enabling technologies especially Web services and Service Oriented Architecture. The chapter also highlights the issue of interoperability among cloud software services. The topics describing the genesis of the problem, the objectives of research, the approach followed, the contributions of research and finally the layout of the thesis have also been included in this chapter.

**Chapter 2** illustrates the models used to describe the system at different levels of abstraction with the help of a Cloud SaaS example. The CIM is represented using the
Use Case diagram and Activity diagram in UML. The PIM is depicted using the UML Class diagrams. The various classes and associations in the class diagram are discussed in detail.

Figure 1.6 Layout of Thesis

Chapter 3 describes the basic concepts of EJB Component model and Relational model. The EJB PSM and Relational PSM are depicted by means of illustrative diagrams. It specifies the transformation definitions for transforming the PIM of the Cloud SaaS example under consideration into an EJB PSM and a Relational PSM, respectively.

Chapter 4 highlights the interoperability issue, and discusses the significance of SOA and Web Services in achieving this goal.
Chapter 5 presents the models, PIM and PSM, of cloud software service interface. It describes the various elements of WSDL2.0. The chapter also presents a WSDL2.0 metamodel which is represented pictorially using a UML Class diagram. Besides, a transformation specification for transforming a PIM of a cloud software service into its WSDL PSM is also described briefly.

Chapter 6 draws a conclusion of the research work conducted by the authors, and the limitations and the future scope of their work.