CHAPTER 5
Approaches for Ontology Based Reusability

5.1 Introduction

Currently, the absolute prospective of software reuse in highly variable domains cause rigid to unleash the software applications. Also, many business concepts constrained by tight regulations are difficult to protract. Building an application from scratch is a resource intensive process for information system, passing by domain-specific variability. When dealing with legacy systems, the cause emerges especially due to the time elapsed between the requirements specification for each developed module and the present. Usually, each application has its own configuration, stores its own data and is guided by its own business rules [CM07]. As a result, during software development, the reuse prospect is generally expected condition [RAM05, FK05].

In addition, it is extensively reckoned that the development and utilization of reusable software artifacts is necessary for improving software development efficiency and software prominence. Most software development methodologies recognize the utility of reuse, and some even provide processes and contrivances to directly support it. Effective software reuse requires collections of designed-for-reuse software components. In addition, mechanisms to retrieve reuse candidates to adapt and create new ones using the information provided by similar components [MER+03]. Moreover, it is needed to bind these elements using a software process that truly accede to software reuse. In this context, ontologies can play an important role. Ontologies have become an important mechanism for building software, since these can be used to overcome barriers created by disparate vocabularies, representations and tools. Ontology may take a variety of forms, but necessarily it includes a vocabulary of terms, and some specification of
related meaning [MF03, OVR+06]. This includes definitions of concepts that are inter-related which collectively impose a structure on domain and constrain the possible interpretations of terms. Attempts have been made to reconcile the terms with feature modelling, domain modelling, etc. However, there is strong need of combining the conceptions of domain with stronger extensibility and with indexing knowledge population. Accordingly, we develop ontology based approaches for reusability. The role of ontologies is to capture domain knowledge in a generic way and to provide a commonly agreed upon understanding of a domain.

In view of this, we introduce Ontop4ViewReuse framework with ontology validated composition. It caters in highly variable domains due to emergence of several dimensions of software development in the course of various abstraction levels. It is based on ontology oriented systematic P4View approach for reusing. Next, OntoReuseAlgo for knowledge integration and reuse towards process planning in software development is commenced. It is based on Ontological Knowledge Modelling to provide reusable and shareable engineering applications. Lastly, we develop Ontological reuse (OnR) from Object-Oriented Reuse (OOR). It potentially applies all the phases of OOR such as development of reusable artefacts, representation and classification of artefacts into repositories, and utilization of the artefacts from repositories. Also, a range of classes of reuse have been identified for comparison of OOR and OnR.

5.2 Background

There exists software reuse around for years and involves variety of concepts. Early software reuse practices focused on code and implemented in adhoc or opportunistic manner. Also, the active areas of reuse research in the past twenty years include domain engineering methods, reuse design, design patterns, domain specific software architecture and component. All these areas well catered by Object-Oriented Reuse [DEV02, SS03, HL01, GH95]. Consequently, reuse has become planned and systematic. But, ontology is adapted to enhance Object-
Oriented Reuse. Ontology is a formal explicit description of concepts in a domain of discourse and oriented towards a systematic method for reusing. In this view, the reuse approach is introduced early in the life-cycle of software development. Accordingly, any product of the software life cycle can potentially be reused. It is a formal and well-documented process which is no more domain-specific and can be recreated. This approach follows a well-planned, lucrative, and productive strategy. In addition, it allows the use of existing software or software knowledge to construct new software. In this section, we describe the subclasses of reuse followed by Object-Oriented Reuse process and then Ontological Reuse process.

5.2.1 Reuse Subclasses

Ontologies have great potential to deal with software reuse predicaments of various aspects such as domain specificity, fixed functionality, well bounded interfaces, performance expectations, and demonstrable excellence [HAM04, TA00]. Consequently, we have identified and explained range of subclasses of reuse as follows:

Software Component Reuse

Software component reuse is the software engineering practice of creating new software applications from existing components, rather than designing and building them from scratch. Reusable components can be requirements specifications, design documents, source code, user interfaces, user documentation, or any other items associated with software. All products resulting from SDLC have the potential for reuse. The practice of component reuse supports the motivation for development of customized applications. Its benefit includes reduced application development time, reduced application cost, and improved application quality [KUH98].

Software Architecture/ Design Reuse

The reusability of software design and software architecture refers to the re-application of representations of one system or component to the construction of similar ones in a problem domain. It is observed that reusability can be
enhanced if the software design and software architecture are explicitly represented and if the representation can be easily understood and manipulated (modified and reconstructed) towards a varieties of target systems [SS99].

Software Requirements Reuse

Requirements reuse is an approach to systematically use existent requirements documents for reducing the general effort inside the software lifecycle. From the point of view of improving requirements engineering, requirements reuse aids by recording the adopted suppositions, made decisions, and adopted alternatives for future reference. It provides the ease of the change management of requirements. Moreover, requirements reuse is helpful in the assistance, guidance and advising for the requirements engineer in the process of requirements acquisition [OM99].

Software Process Reuse

Software process reuse represents a new practice for software production in which a conceptual knowledge representation is used to represent and guide development activities. During software process reuse, process engineers specify a software process that is tailored for project goals and other resource constraints, and then enact the process as a guide for developers [HOL98].

Software Technology Reuse

Software technology reuse provides certain types of services to their users such as storage, searching, inspecting and retrieval of artifacts from different application domains, and of varying granularity and abstraction, loading, linking and invoking of stored artifacts, and specifying artifact relationships [BAR06].

Software Experience Reuse

Software experience reuse enables people to effectively reuse components. It is observed that, the visual interface design is perhaps even more important than the user need as a succinct way of communicating the purpose of the component to designers [ABH+99].
5.2.2 Object-Oriented Reuse Process

To effectuate reuse, three major engineering activities must be addressed as shown in Figure 5.1. Firstly, reusable artifacts must be intentionally designed and developed. Secondly, reusable artifacts must be represented, classified, and entered into and removed from appropriate repositories. And, subsequently tools and processes must be developed that support finding, understanding, modifying, and composing artifacts [WER97, SMJ02]. Now we discuss them as follows:

Development of Reusable artifacts

Development of reusable artifacts concerns with the work required to establish a set of software artifacts that can be reused by the software engineer. Its purpose is to identify, model, construct, catalog and disseminate a set of software artifacts that can be applied to existing and future software in a particular application domain.

Representing Reusable artifacts

The most difficult problem with reuse is developing a suitable representation for artifacts. In particular, it resembles a representation that encodes the semantics of artifacts. Users trying to solve a problem with own knowledge and semantics can locate an appropriate reusable artifact. Such artifacts must be retrievable by multiple pathways to support variety of different ways in which users may access them. Furthermore, representation allows for a variety of different perspectives on stored artifacts, and permit versioning and configuration management activities. Also, it allows for representation of partial and uncertain information. This allows artifact developers to evolve the designs over time by permitting well-defined aspects to be expressed with certainty, and less well-defined aspects to be left fuzzy.

Repository Reusable artifacts

Artifacts must be classified and entered into repositories, once artifacts have been represented. Classification of artifacts is an indexing issue. As such, artifacts are classified in order to indicate the type and relation to other artifacts.
There exist two well known schemes for repository classification accomplishment such as enumerative and faceted. Enumerative scheme divides the universe into a collection of domains and sub domains. However, faceted approach does not rely on a prior division of the universe into domains, but rather synthesizes a classification of an artifact based on the selection of properties from a collection of facets.

Supporting the Reuse of artifacts

Once artifacts have been developed, represented, and categorized into repositories, the next concern is to utilize this wealth of information. Software developers need tools and processes for finding, understanding and using reusable artifacts.

Finding artifacts

To find artifacts, users describe the requirements and tools included for requirement satisfaction. This simple declarative model is rarely achieved in practice as most representations are insufficient to support sophisticated queries and reasoning. Sophistication of techniques for finding information is dictated by the representation scheme. Hence, the extent of OOSE to support retrieval will be dictated by representation scheme.

Understanding artifacts

Once artifacts have been located, it is necessary to understand in order to use these artifacts. OOSE has potential to enhance the understandability of software artifacts. Strength of an object-oriented approach is that it offers a mechanism that captures a model of the real world termed as objects.

Using artifacts

This activity has been viewed as a fundamental part of development process. There exist varieties of different ways in which an artifact may be reused. A retrieved artifact that is useful without modification need only be integrated i.e. "plugged" into the system. However, if an artifact requires modification, it may be
Figure 5.1: Object-Oriented Reuse Process
necessary to refine or compose it. Then, combination of retrieved artifacts is
required. In an object-oriented system, the refinement and composition tasks are
potentially simpler. By using inheritance, refinement is described as top down
process of specifying the differences between inherited state and behavior of an
existing object and requirements of desired object. With excellent support of
encapsulation and message protocols, composition is a bottom up
process of connecting together the proper object building blocks to
form the desired component [SS04].

5.2.3 Ontological Reuse Process

Ontological reuse process starts with the identification of knowledge
sources useful for the application domain that differs in represented content as
well as in the formalization [SMJ02]. An automatic integration of the source
knowledge does not mean only the translation of the representation languages to a
common format, but also the matching of the resulting schemes. Ontological
reuse process has been introduced early in the life-cycle of software development
as it is a formal and well-documented process which is domain unambiguous and
can be recreated as shown in Figure 5.2. The process is describes as follows:

Determine Scope

It refers to defining concepts in the domain (classes). There exists no
correct ontology of a specific domain. Ontology is an abstraction of a particular
domain, and there always subsists viable alternatives. This abstraction must
be determined by the use to which the ontology kept and by future extensions
that are already anticipated.

Define Taxonomy

It ensures arranging the concepts in a hierarchy (subclass super class
hierarchy). Since, the hierarchy must be efficient or reliable hence user opinions
may differ to select the type of hierarchy to define taxonomy. The types include
top-down or a bottom-up fashion.
Figure 5.2: Ontological Reuse Process
Define Properties

It defines attributes or properties (slots) that classes can have and constraints on their values. While attaching properties to classes, it has been observed that it immediately provide statements about the domain and range of these properties. There exists a methodological apprehension between generality and specificity such as flexibility (inheritance to subclasses) and detection of inconsistencies and misconceptions.

Define Facets

It asserts defining individuals and filling in slot values with cardinality restrictions and relational characteristics such as symmetry, transitivity, inverse properties, and functional values.

Define Instances

Filling the ontologies with instances is a step concerned with creating a knowledge base. It defines individual instances of various classes and filling in specific property such as specific slot value information and additional slot restrictions.

5.3 P4View Approach Based Framework

The key challenge while managing and characterizing reusability in highly dynamic domains is to identify the relations and representations of software artifacts and resources involved. In such context, we have proposed P4View approach for building systems, adaptable to each user with common characteristics. This approach makes use of ontologies pact to the knowledge and experience of users, history of previous actions, goals, intentions, interests and preferences. Using this approach, OntoP4ViewReuse framework for software development is described. Framework emphasizes on different levels of abstraction that provides an unambiguous terminology, allowing its reuse and easy extension. The detailed description of P4View approach and OntoP4ViewReuse framework is as follows:
5.3.1 P4View Approach

We have proposed a P4View that resorts to available ontological knowledge and are implicitly tailored to specific application needs. In turn, it cannot be reused in different settings. While, in P4View approach additional ontological primitives like properties and axioms are supported explicitly. P4 stands for Pretence-Persuade-Problem-Product, defining the various abstraction levels to be accomplished during the software development. Figure 5.3 represents the overview of P4View approach, references ontologies. Each of these views represents a meticulous attribute of the ontology and defined as follows:

Pretence View

Pretence view caters by high level ontology that includes representational, terminological and social ontologies. Representational ontology helps in identification of knowledge sources useful for the application domains that differ both in represented content and in formalization. In addition, terminological ontology describes general concepts that are independent of a specific domain or a problem such as space, material, objects etc. Lastly, social ontology includes the terms such as actor, position, role, authority, responsibility or commitment.

Problem View

An automatic translation of the source ontologies from a common format to the representation languages is carried out at Problem view. It is supported by domain ontology that is comprised of informational, intentional and static ontology. Informational ontology structures the standardized storage of information. While, intentional ontology describes aspects of world of intentions, goals, beliefs alternatives and elections of involved users. Lastly, static ontology describes the terms such as entity, object and relationship.

Persuade View

The identification of terms specific to the problem resolution methods and or tasks is involved at Persuade view. This view includes dynamic and method
Figure 5.3 P4View Approach
ontology. Dynamic ontology articulates terms such as process, state or state transitions. In addition, matching of the ensuing method is accepted at *Persuade view*. Hence, task ontology signifies method ontology. It offers a reasonable point of view to the knowledge of the domain.

**Product View**

It categorizes the roles played by the domain entities when executing an activity. It includes the controlled vocabularies, informal and formal hierarchies, frames, value constraints and generic logical constraints prolonged with application ontology. Finally, application ontologies revealed the reuse source vocabularies to a large extent in *Product view*.

**5.3.2 OntoP4ViewReuse Framework**

*OntoP4ViewReuse* is based on ontology oriented systematic *P4View* approach for reusing. *OntoP4ViewReuse* bring about to apply the ontology of varying levels of notion such as high level, domain, task and application ontology. This cataloging of ontologies is useful for the development of reusable and high-quality software application. In addition, through ontologies, the eliciting and modelling of the knowledge is being carried out using *P4View* approach that concentrates on different levels of abstraction. Initially, the general knowledge of the domain is elicited and specified in one or more views and finally serves the next views to develop the specified application. As a result, phase wise procedure is introduced to construct *OntoP4ViewReuse* framework. These phases are described as below:

**High Level Ontology Phase**

We have consider all possible aspects of a system such as its type, associated directives and activities performed by people in various types of system with its own rules, to define the scope of this ontology phase. As depicted in Figure 5.4, *System* is composed of *System’s Type* in which different *Activities* are performed by *Human Resources*. We also include the fact
Figure 5.4 *Pretence View using High level Ontology*
that a system adopts *Directives* to be followed in the execution of the tasks. Database management systems, utilities and system softwares such as operational, network and middleware are included in *System’s Type*.

*Activities* are majorly classified into three types such as investigation, modification and management. *Activity* uses one or more input artifacts and affects one or more output artifacts. It precedes some activity and or part of some other activity. Investigation activities focus on assessing the impact of undertaking the modification where as management activity contributes to the configuration control of the products. In addition, modification activity includes corrective and enhancement activity aiming at adaptive, preventive and perfective continuation during the product construction.

*Directives* support system such as online documentation to help and contrivance guidelines, Architectural design for dynamic library reuse and Requirement for change to specify the new one. Also, data structures such as structure of data files or databases are mentioned with Interoperability to feature the communication with other systems and Security to ensure the integrity of system. Finally, execution of system is included for performance or instability measurements.

*Human Resources* include software engineers such as suppliers and maintainers. Supplier develops the system and maintainer maintains the system. In addition, maintenance manager is responsible to conduct concerned maintenance procedures and client human resources include clients and users.

**Domain Ontology Phase**

In the context of software engineering a domain defines as an application area, for which software system has to be developed. Domain Ontology refers to reuse-based process used to define scope and structure. Also, it illustrates reusable attribute for various kinds of system having different domain specifications and support specifications. Taxonomy of domain can be decomposed into its *Job* and *Components* as shown in Figure 5.5. It consider and represent similarities and
Figure 5.5 Problem View using Domain Ontology
difference between the systems within a domain. *Components* represent all the coded artifacts that compose the software program itself. These are classified into execution components generated for the software execution and deployment component for composing the executable program.

*Job* is decomposed into two kinds according to the type of specifications such as domain specifications and support specifications. Domain specifications are composed of requirement, design and product specifications for describing the system's behavior and structure. Different view models may be defined to redefine the design specifications at logical and physical level. Moreover, support specifications helps in operating the system such as document to illustrate the results obtained from the study of the reuse of requirements, software design, and generic architectures. Also, it includes identification of hardware to install the system and the compatibility of software with it. In addition, model illustrates information in an understandable fashion through formal presentation.

**Task Ontology Phase**

Tasks or procedures are the structured descriptions used in a software development activity such as *Methods*, *Techniques* and *Assertions* as shown in Figure 5.6. *Methods* are the kind of systematic procedures with semantic and syntactic definition to be followed. On the other hand, *Techniques* are the logical procedures less formal and rigorous than a method. *Techniques* begin with requirement elicitation that includes procedures (such as interviews and brainstorming etc.) to assist in the identification of requirements. Subsequently, modelling techniques adopts specific modelling language to define the systematic solution for a problem followed by programming technique (may be structured or object oriented). Consequently, testing techniques include such as white or black box etc. Lastly, maintenance techniques classified into reverse engineering, re engineering, impact Analysis and program comprehension to assist in the maintenance of program. Lastly, *Assertions* defines directives or the standards such as guidelines or norms defined to use the system.
Figure 5.6 Persuade View using Task Ontology
Application Ontology Phase

Application Ontology organizes the process that builds products from software elements abstracted through domain and task ontology. Application ontology depicts Concept and Task that compose an application. Also, Properties associated with each Concept and Restrictions applied to an application is specified as illustrated in Figure 5.7. Concept is aiming at satisfying the application needs of a specific kind of user and performance expectations. Next, Property refers to a component that supply the functionality needed by the user. And, Restrictions signify to constraints that may be logical or value applied during the software development to validate the requirements. Lastly, Procedures indicate functionality of product that must be fixed, along with its preconditions and post-conditions. Thus, users will know exactly the product’s function under all circumstances.

Now, we integrate all these phases in a single conceptual framework OntoP4ViewReuse as shown in Figure 5.8. The framework contemplates on different levels of abstraction namely, Pretence, Problem, Persuade and Product views consistent with types of defined ontologies. Levels of abstraction relate to the completeness, and to the value of reusable property. Level 1 abstraction signifies the constituents as agreed for repository population on the basis of generalized stipulate only. This level established with the components of high level ontology such as representational, terminological and social ontology. Representational ontology indicates system, system’s type. Next, Activities and directives present terminological ontology and human resources are explicitly defined inside social ontology. The completeness of the high level ontology components is well recognized at this level.

Subsequently. Level 2 abstraction configures domain ontology constituents such as informational, intentional and statical ontologies. Informational ontology defines the components. Statical ontology describes domain specifications and intentional ontology indicates support specifications. As discussed earlier, these specifications are included in Job constituent.
Figure 5.7: *Product View using Application Ontology*
Next, Level 3 abstraction renders task ontology. This level determines many idiosyncratic such as methods, techniques and assertions that are perceptibly delineate in methodical and dynamic ontologies of task ontology. Lastly, Level 4 abstraction structures the application ontology constituents such as vocabularies, hierarchies (formal and informal), frames and constraints (may be logical and or value) that is to be released to users and it verifies completeness. A concept that is part of property associated with task and restrictions offer a highest degree of abstraction.

5.3.3 Case Study

To tap the full potential of existing domain-relevant knowledge sources, ontology is being accepted. At this instant, reuse with the help of ontology is defined as the process in which ontological knowledge is used as input to generate new ontologies. Depending on the content of the knowledge sources and domain overlapping, the implications of reuse in the overall development process can be clarified. We address the reuse process to a greatest extent using OntoP4ViewReuse framework in the domains of e-Recruitment and e-Medicine.

Case I- e-Recruitment portal

e-Recruitment portal allows a uniform representation of job postings, job seeker profiles and semantic matching in job seeking and procurement tasks. It facilitates to support common practices from the industry and to maximize the integration of job seeker profiles and job postings from different organizations. High level ontology underlying this job portal is aligned to established domain-specific standards and classifications. The selection of high level ontologies is followed by the customization and integration to the new ontology. We identified the sub-domains of this system such as networked system type includes professional, educational and industrial areas. Next, domain ontology is used to define concepts representing competencies to describe job requirements as well as job seeker skills. Due to the domain setting, component classification standards such as the occupation component and the industrial sectors component have to be completely integrated in the new ontology. To extract the relevant fragments from
task ontology, we compiled a small conceptual vocabulary from various job portals and job procurement web sites and matched these core concepts to the source ontology. The usage of the ontology in semantic matching tasks requires that it is represented in a highly formal representation language. For this reason the implementation of new ontology has realized by translating several semi-structured input formalisms using application ontology.

**Case II- e-Medicine portal**

We developed e-Medicine portal for lung pathology to analyze the practice in a retrieval system for representation content data in the medical domain. This e-Medicine portal provides a concept-based reuse technique and semantic annotation of pathology reports. To develop high level ontology, we identified anatomical, clinical and pathology-specific system type and separate the application relevant knowledge from the general purpose medical knowledge.

On the other hand, domain ontology covers both domain and application-relevant knowledge that is specific to the health-care institution involved in the project. Also, medical components such as digital anatomist are tailored to domain ontology. For this purpose domain experts identified four central concepts such as “lung”, “pleura”, “trachea” and “bronchia” and included to the task ontology. Also, this standard format for the representation of patient data and patient records and immunohistology guidelines used by domain experts in diagnosis procedures, significant parts of the pathology domain, are integrated in task ontologies.

A large part of pathology specific method such as vocabulary with a lexicon generated from an archive of medical reports resulted in further refinements of the application ontology. It is implemented to describe the anatomy of typical diseases aligned to generic and core medical concepts. Additionally, application ontology is needed for semantic annotation required a maximal coverage of the vocabulary used by domain experts in medical reports.
5.3.4 Benefits of OntoP4ViewReuse

We propose OntoP4ViewReuse framework using P4View approach to utilize the content of the source ontologies to a maximal extent depending on their particular domain and level of formality. Adopting OntoP4ViewReuse based software development process attracts a number of benefits to both the end-users and developers. These include the following:

**Savings in costs and time:** As a developer uses already pre-defined components, hence, the activities associated with components specification, design and implementation are now replaced with finding components, adaptation to suit new requirements, and their integration. Though, ontology based reuse certainly attract additional effort, time and cost. These costs, however, can be offset by savings in a number of different software projects.

**Increase in productivity:** It has been shown that reusable artefacts developed from OntoP4ViewReuse can be viewed as abstract level of concepts drawn from a given problem domain. Hence, working with such higher level of abstraction leads to an increase in development productivity.

**Increase in ease of maintenance:** Systems constructed of reusable parts are usually simpler and more abstract. Also, the designs are closer to the problem domain and their consistency is greater. This of course has very positive impact on the quality of such systems maintenance.

**Increase in reliability:** OntoP4ViewReuse suggests that the life-span of reuse artifacts is much greater than that of any individual product. Thus, the reliability of such artifact is also increased. This also leads to an improved reliability of systems built of reusable components rather than of those built entirely from scratch.

**High speed and low cost replacement of aging systems:** Systems developed using OntoP4ViewReuse shares a very large collection of concepts via ontology, thus, have become significantly multifaceted. Such systems need less effort during porting or adaptation to new hardware software environments. Also, the
reusable components of the system are technology intensive and very expensive to develop but sharing that cost across several systems certainly reduce it when a global replacement of computing resources have effect.

5.4 Ontological Knowledge Modelling Based Algorithm

Reuse also responds to an increasing insist for highly reliable, high excellence and less expensive systems. Accordingly, knowledge reuse benefits and improves the process planning in software development greatly. Process planning is an intermediate phase between design and implementation. Lucidity and prescribed specification of concepts play a key role in the inclusion of reuse during process planning. Therefore, a most important issue is to build a common conceptual base characterized by knowledge. Our exploration focuses on this task through developing OntoReuseAlgo based on Ontological Knowledge Modelling. The brief description of Ontological Knowledge Modelling is illustrated below:

5.4.1 Ontological Knowledge Modelling

We propose an Ontological Knowledge Modelling for knowledge integration and reuse towards process planning in software development. It constitutes System Element Classification, Ontolayering Principle and Knowledge Reuse Scheme to provide reusable and shareable engineering applications. The detailed description of these constituents is as follows:

System Element Classification

It is developed to capture important characteristics for reducing the growing complexity of information and increasing need to exchange it among various software applications. The classification includes abstract concepts such as Work units, Stages, Work products and Producer as shown in Figure 5.9. Work units constitute tasks or activities that software developers perform, and have a start and end time as well as duration. Subsequently, Stages describes major time frames that help work to provide temporal structure.
Figure 5.9: Classification of System Elements
Next, Work products such as documents or software, are intangible results of performing work units indicates creations and last change times with status. The status of work product may be initial, complete, accepted or approved. Finally, Producers includes people and teams that actually perform work units in order to create work products.

**Ontolayering Principle**

*Ontolayering Principle* focuses the ontology in a resource usage manner, specifically by understanding and dissimilating the information comprised by entities. The three prospects namely; Metamodel, Process and Product prospects have been defined around communities that network with ontology as shown in Figure 5.10. Meta-model prospect acts as a common standard determining the other prospect. Meta-model is intended to be used as an origin by method engineers so that the methodologies can be developed. Method engineers typically uses the concepts in meta-model prospect by sub typing and instantiation, thereby creating new concepts (subtype of existing ones) and entities (instances of concepts). All these new concepts and entities created by method engineers are seized to form a Process prospect. Software developers use it by creating the instances of concepts and also, by following the guidance explained by entities. Thus, the instances created by software developers are apprehended to form Product prospect.

**Knowledge Reuse Scheme**

It starts with formalizing the system element requirements according to representation approach of *System Element Classification*. Subsequently, identification of the related process concepts and entities that need to be revised according to *Ontolayering Principle* retrieved from the knowledge base. Consequently, modification of the producer entities, associated concepts based on *Ontolayering principle* and revision of the influencing product attributes conceded. Finally, simulation of the results is done if the reuse requirements are satisfied otherwise it considers changing a different process concept and entities.
Figure 5.10: Architecture of Ontolayering Principle
5.4.2 Ontology Based Reuse Algorithm (OntoReuseAlgo)

We propose Ontology Based Reuse Algorithm (OntoReuseAlgo) using Ontological Knowledge Modelling approach to aid the product design of process plans. It is used to give a uniform representation of the involved information and starts with understanding the system elements. It includes identifying process concepts and entities that need to change followed by altering with Ontolayering principle and modify the producer entity and the associated concepts which help in revising the influencing product attributes for simulating the final process plan. Figure 5.11 shows overall procedure of the proposed approach and knowledge reuse strategy in process planning. The stepwise procedure of process planning task is as depicted:

**Step I:** Formalize the system elements according to the representation approach of System Element Classification.

**Step II:** Identify the related process concepts and entities that need to be added according to Ontolayering principle retrieved from the knowledge base.

**Step III:** Modify the producer attributes and associated concepts based on Ontolayering principle.

**Step IV:** Revise the influencing work product.

**Step V:** Simulation of the results in step IV. //if the knowledge reuse is satisfied, then shifts to step V; or else, shift to step II and consider changing a different process concept and entities.//

Different systems may use different concepts and terminology to express the same objective while same words may be used to represent different objective by different systems. Both situations hinder information communication. Therefore, step I uses System Element Classification. Conversely, process and product concepts that need to change are properly identified and revised through mappings of related Ontolayering Principle to certain process and product prospects in step II, III and IV. Lastly, Knowledge Reuse Scheme is applied to step V for providing vocabulary and the meaning of the terminology.
Figure 5.11: Ontology Based Reuse Algorithm (OntoReuseAlgo)
5.4.3 Case Study

We describe design of process planning for Inter-warehouse Management System using OntoReuseAlgo. The system is responsible for redistribution between different warehouses. The window for redistribution between warehouses is as shown in Figure 5.12. Various people are responsible for carrying out different processes such as foreman is responsible for warehouse management. While, warehouse worker works in a warehouse for loading and unloading. Subsequently, truck driver is accountable for transportation and forklift operator drives a forklift in one warehouse.

Now, on executing first step of OntoReuseAlgo, we analyze various system elements such as work units that include request for redistribution, fetching item from warehouse and delivers the item to the new warehouse. Then, producers constitute foreman, warehouse worker, truck driver and forklift operator. Consequently, work product comprised of initialization, loading and unloading as illustrated in Table 5.1. While, during the execution of second step, we identified various process concepts that need to be added. These processes include unexecutable request, wrong redistribution and unavailability of truck as depicted in Table 5.2. Accordingly, step three suggests modification of producer element of step one by including office personnel in it. Official personnel coordinate the transport requests that affect initialization, loading and unloading. Finally, execution of step four recommends a new influencing product attribute termed as planning as shown in Table 5.3. Therefore, revised work product comprised of initialization, planning, loading and unloading.

Lastly, we simulate work product that starts with completion of initialization work unit than planning, loading and unloading. This revised work product eases the redistribution between the warehouses.
## REDISTRIBUTION BETWEEN WAREHOUSES

<table>
<thead>
<tr>
<th>Items</th>
<th>From Place</th>
<th>To warehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screww6&quot;</td>
<td>A12</td>
<td>Aivesta</td>
</tr>
<tr>
<td>Oil Drum</td>
<td>A15</td>
<td>Stockholm</td>
</tr>
<tr>
<td>Computers</td>
<td>D32</td>
<td>Lund</td>
</tr>
<tr>
<td>Bananas</td>
<td></td>
<td>Kalmar</td>
</tr>
</tbody>
</table>

**Redistribution No:** 123456

<table>
<thead>
<tr>
<th>Item</th>
<th>From</th>
<th>To</th>
<th>Quantity</th>
<th>When</th>
</tr>
</thead>
<tbody>
<tr>
<td>bananas</td>
<td>A15</td>
<td>Lund</td>
<td>All</td>
<td>920315</td>
</tr>
</tbody>
</table>

Figure: 5. 12: Redistribution between Warehouses Window
<table>
<thead>
<tr>
<th>S. No.</th>
<th>Work Units</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1 | **Initialization**<br>(when foreman gives request to do the redistribution) | 1. The foreman gives a command for redistribution between warehouses.  
2. The window in Figure 5.12 is presented to the foreman.  
3. The items can be ordered in a number of ways with ORDER menu such as alphabetical, index, turnover of the items and storing order.  
4. In the ‘From place’ table we may choose to view either all places in the current warehouse or, if we have selected an item, the place where the item exists.  
5. In the ‘To warehouse’ table we may select all warehouses or the warehouses that we have to transport to this week.  
6. The ‘Issuer’ and ‘warehouse’ fields are automatically filled when the window pops up.  
7. The foreman selects an item by pointing to it and dragging it to the Redistribution form then selects from which place to take the items and to which warehouse to transport them.  
8. The foreman then gives the quantity to be moved and the date.  
9. It is possible to change the information when the form has been edited. When the foreman EXECUTES the redistribution, the transport is planned. It is also possible to CANCEL the redistribution. Selecting HELP shows window of information about the current window. |
| 2 | **Loading**<br>(when truck fetches the item from the warehouse) | 1. A Truck driver asks for a transportation request. The request is marked as ongoing.  
2. Give an appropriate request to the Forklift operators to have the items ready when and where the truck is expected.  
3. When the Warehouse Worker gets a request to fetch items at appropriate time, orders Forklift operators to move the items to the loading platform.  
4. When the Truck driver arrives the items are loaded. The Truck driver tells the system when the truck is loaded and when it is expected to be at the new warehouse.  
5. Decrease the number of items in this warehouse and mark the transport request as on transport. |
| 3 | **Unloading**<br>(when a truck delivers the items to the new warehouse) | 1. When the truck has arrived at the new warehouse, the items are unloaded.  
2. The Truck driver tells the system that the transport to this warehouse has been done.  
3. The Warehouse workers receive the items and determine a place for them in the warehouse.  
4. Forklift operators are told to move the items to the new place in the new warehouse.  
5. When the Truck driver confirms the insertion, the system updates the new place for the items.  
6. The transportation time is recorded and stored in the system.  
7. The Redistribution and the transport request are marked as performed. |
Table 5.2: Identified Processes

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A request is not executable</td>
<td>The execution is interrupted and the Foreman issuing the request is informed</td>
</tr>
<tr>
<td>2</td>
<td>Redistribution is wrong</td>
<td>The warehouse place does not have enough items to move. The destination warehouse is not appropriate to the item</td>
</tr>
<tr>
<td>3</td>
<td>No truck available</td>
<td>When performing loading, and unloading, there may not be any truck available at an appropriate time. Then notify the Foreman who should either delete the request or change it.</td>
</tr>
</tbody>
</table>

Table 5.3: Influencing Product Attribute

<table>
<thead>
<tr>
<th>S. No</th>
<th>Planning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To coordinate transports and issue transport requests</td>
<td>1. When the redistribution is executed the items to be moved are marked as move-pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Minimize the use of trucks on condition that all delivery dates should be held and the trucks should be compatible with any delivery requirements for the items.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. The transport requests are connected to a specific truck’s transportation plan.</td>
</tr>
</tbody>
</table>
5.4.4 Benefits of *OntoReuseAlgo*

*OntoReuseAlgo* aims to improve the knowledge reuse in process planning for software development. It supports the application from three aspects such as *System Element Classification*, *Ontolayering Principle* and *Knowledge Reuse Scheme* for process planning. We have observed the following significant benefits:

- Through organizing and modelling the knowledge towards the characteristics of design processes, unnecessary search time can be avoided on irrelevant knowledge items.
- It allows explicit credentials for analysis and comparison of different domain theories.
- It describes knowledge acquisition approach to structure the entities and relations that need to be acquired in the domain.
- It provides a meta-level view (vocabulary and structure) on their domain which facilitates adequate system documentation and constructs reusable knowledge-system design.
- It can be used to define assumptions that enable knowledge exchange between different users.

5.5 Ontological Reuse (OnR) from O-O Reuse(OOR)

An available reuse methodology such as OOR addresses reusability issue only marginally. Though it mentions the possibility of reusing existing knowledge sources as input for the conceptualization phase, it fails to define precisely knowledge discovery and the subsequent evaluation of candidate knowledge. Also, it describes in detail to build and represent reusable artifacts, but furnish a relatively sketchy recommendation for supporting existing reusable artifacts. Figure 5.13 illustrates more pragmatic process OnR, which exploit OOR process to a maximal extent depending on the particular domain and level of formality. OnR process extremely addresses this issue in the context of knowledge
customization/pruning, explicitly extracting relevant fragments from very comprehensive, general purpose ontologies [SS99, REI97]. In addition, OnR provides a detailed description of reuse process and its implications in the overall engineering process.

5.5.1 Mechanism of OnR Development

We propose a generic and incremental process that concentrates on vocabulary of the input sources. And, subsequently inserts additional information corresponding to application needs. OnR process taps the full potential of OOR process from development and representation of reusable artifacts to supporting the reusable artifacts. Figure 5.13 illustrates the mapping of OOR notions analogous to each of OnR notions. The development and representation of reusable artifacts of OOR are concerned with the identification, modelling, cataloging and disseminating a set of software artifacts that can be applied to existing and future software in a particular application domain.

Besides, it encodes the semantics of artifacts in such a way that a user, trying to solve a problem with own knowledge and semantics can locate an appropriate reusable artifact. Therefore, these notions mapped with determining the scope, taxonomy and properties. In OOR, once artifacts are represented, they must be classified and entered into repositories. As such, artifacts are classified in order to indicate the type and relation to other artifacts. There are two well known schemes for doing this repository classification: enumerative and faceted [KG02]. Formerly artifacts have been developed, represented and categorized into repositories; the next concern is to utilize this wealth of information specifically supporting the reusable artifacts. Thus, relates with defining the facets and instances of OnR.

5.5.2 Categorical Comparison of OOR and OnR

On the basis of aforementioned reuse subclasses, we have presented a comparative study to systematically exemplify the object oriented reuse versus ontological reuse as shown in Table 5.4. To begin with, Software Component
Figure 5.13: Object Oriented Reuse to Ontological Reuse
Table 5.4: Object Oriented Reuse vs. Ontological Reuse (Contd ...)

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Reuse Subclasses</th>
<th>Object oriented Reuse</th>
<th>Ontological Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Software component reuse</td>
<td>• Reduce the number of parameters.                                                   • Enables multi faced description of components by lexically analyzed, stored, and indexed using the tokenization and indexing mechanisms.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Avoid using options in constraints.                                                • Generating semantic instances for the concepts and relations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Prohibit the direct access to instances.                                            • Allows semantic matching based upon signature-based queries and metadata keyword queries.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Implies more cohesive and primitive operations.                                     • Structures knowledge acquisition for entities and relations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Identify the system’s responsibilities at a given level of abstraction.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Scrutinize the system’s environment to produce classes and objects.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Partitioning the class and the object structure into larger units for various attributes and services.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Identify generalization-specialization structure to capture inheritance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enables analysis and comparison of different domain theories.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software architecture/design reuse</td>
<td>• Structures knowledge acquisition for entities and relations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Provide a metalevel view (vocabulary and structure)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Facilitates adequate system documentation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Define assumptions that enable knowledge exchange between different agents.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Software requirement reuse</td>
<td>• Do domain analysis and prototyping.                                                 • Defines the optional parts of candidate system.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Perform incremental development.                                                    • Models the complex and alternative courses which seldom occur.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reconcile conflicting sets of requirements.                                         • Outline separate sub courses which are executed only in certain cases.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Build flexibility by regular reviews.                                               • Models the situation in which different modes can be inserted.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td>• Allows analysis and comparison of different domain theories.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Structures knowledge acquisition for entities and relations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Provide a metalevel view (vocabulary and structure)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Facilitates adequate system documentation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Define assumptions that enable knowledge exchange between different agents.</td>
<td></td>
</tr>
<tr>
<td>S. No.</td>
<td>Reuse Subclasses</td>
<td>Object oriented Reuse</td>
<td>Ontological Reuse</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4.</td>
<td>Software process reuse</td>
<td>• Recognition of objects and operations.</td>
<td>• Generate implementation for the Knowledge Processor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Grouping the objects and operations.</td>
<td>• Import /export model elements from /to the Repository.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Presents the pictographic description of objects.</td>
<td>• Generate a generic Task from the Ontology.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Justifying the design decisions.</td>
<td>• Find an implementation from a Repository.</td>
</tr>
<tr>
<td>5.</td>
<td>Software technology reuse</td>
<td>• Provides diagram of each class hierarchy and collaboration for each subsystem.</td>
<td>• Based on open-source components and will be published as open-source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Specifies contracts supported by each class and subsystem.</td>
<td>• Support interoperability, and standard-based solutions are preferred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Presents computational description of each public object module.</td>
<td>• Readymade Tool-level extensibility.</td>
</tr>
<tr>
<td>6.</td>
<td>Software experience reuse</td>
<td>• Specify a set of basic building blocks for constructing primitive terms.</td>
<td>• Automating the process of creating application ontologies.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Define the set of consistent symbol structure states, or changes of states accompanied by interpretation rules and usage guidelines.</td>
<td>• Provide means for visualizing, browsing and exploring domain.</td>
</tr>
</tbody>
</table>
subclass concerned with a search for components that supply the functionality needed by the user. Next, Software Architecture/Design Reuse subclass claims to be more than just component reuse since it is one of the software elements to be reused during the software process. Software Requirements Reuse subclass is about sharing the requirements across various domains. Software Process Reuse is a kind of reuse deals with the construction of reusable software processes as a means of improving the organization’s software process. To investigate the application domains of software technologies the Software Technology Reuse subclass has been devised. Software Experiences Reuse describes the methods that try to reuse every useful experience in software systems development.

5.5.3 Case Study

It is vital to practice OnR in software development for the precise granularity and for a high level of stability as indicated in Table 5.4. OOR practiced on domain level which restrains basic terms of a domain that combined and extended in OnR in order to describe more complex semantics. We consider Hydrology Plant Management System to observe effective use of reuse subclasses during OnR. The system is developed to formalize the concepts associated with it such as water level measurement, water body type and water discharge management. Water level measurement deals with required amount of water to initiate the working of plant and convert the hydro energy into electric energy. It is stated that every water body such as sea, river, pond etc. has a water level and a discharge and these qualities can be observed and managed for its use in plant. This general description provides an entry point for Software Requirement Reuse subclass. While, representation for specific water level measurement service is then sanctified at Software Process Reuse subclass. In particular, sea water level has to be higher than river or pond water level as sea water has more impurities than the others. Therefore, OnR offers a knowledge driven approach for dealing with such diversity.
Once the domain level is settled, components can be added or removed without the need of modifications on domain level which makes the application level highly flexible and consequently referred to Software Component Reuse subclass. The commitment to the same domain level makes OnR comparable with OOR. Also, the task of constructing application lies in the responsibility of the provider of the information source whereas the construction of domain knowledge is a joint effort of domain experts that propose Software Experience Reuse subclass. Additionally, all other peculiarities related to specific water level measurement are described at Software Technology Reuse subclass. This includes legal information to use the provided information and data representation issues. The concepts used to describe the knowledge acquisition and exchange aspects are taken from other types of domain such as measurement or data representation ontologies that constitute Software Architecture/Design Reuse subclass.

As stated earlier that OOR caters at domain level only. Hence, it enables to extend the features of a particular application using various reuse subclasses and thus, restricted to that application only. For example, while developing Hydrology Plant Management System, OOR helps to add water harvest mechanism for utilization of waste water using reuse subclasses. Whereas, OnR caters at inter-domain level and hence enables to develop different applications with the help of various reuse subclasses. For example, Solar Plant Management System can be developed using OnR that converts solar energy into electric energy using the knowledge of Water Plant Management System. It includes the concepts such as solar power level measurement, solar energy source and release management.

5.5.4 Benefits of Ontological Reuse (OnR)

OnR achieves some lucidness of unclear concepts related with software reuse. A significant aspect of OnR suggests its independence from implementations or technological aspects. OnR allocates various software reuse subclasses with ensuing benefits such as:
• **Cost reduction:** It helps to reduce cost in terms of smaller number of software requirements specification, design, implementation and validation.

• **Higher reliability and quality:** OnR provides higher reliability and quality by components that are tested in previously functioning systems and thus are more reliable than new ones.

• **Risk reduction:** OnR reduces the risk factor as previously existing process implies determines lesser degree of uncertainty a with respect to cost estimation for the project.

• **Accelerated system development:** OnR provides software architecture/ or design reuse that facilitates in shorter development and validation times.

• **Effective use of specialists:** Instead of application specialists doing the same work in different projects, OnR helps these specialists to develop software that encapsulates the associated knowledge.
5.6 Summary

In this Chapter, we have highlighted the importance of reusability in software development process. Chapter starts with depiction of existing reuse subclasses followed by introduction of Object Oriented and Ontological Reuse process. As ontology based reuse is an emerging aspect and specially used for resolving scalability and heterogeneity issues. In this view, we have proposed reusable framework OntoP4ViewReuse based on ontology oriented systematic P4View approach for reusing. The necessity of P4View approach is to make available ontological knowledge that is implicitly tailored to specific application needs. OntoP4ViewReuse bring about to apply the ontology of varying levels such as high level, domain, task and application ontology. This cataloging of ontologies is useful for the development of reusable and high-quality software systems. Consequently, we have explored a range of benefits of using OntoP4ViewReuse. In addition, to build a common conceptual base characterized by knowledge, Ontology Based Reuse Algorithm (OntoReuseAlgo) for process planning has been proposed. It has supported the application through system element classification, ontolayering principal and knowledge reuse scheme. Also, the significant benefits of OntoReuseAlgo have been drawn. In addition, Ontological Reuse (OnR) has been devised from Object-Oriented Reuse (OOR) and effectiveness of OnR has been highlighted with comparative study based on software component, architecture, requirement, process, technology and experience reuse subclasses. Lastly, benefits of OnR have been delineated.