Using Software Engineering Metrics In AP Modularization

A thesis

presented to

the faculty of the Department of Computer and Information Sciences

East Tennessee State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Information Science

---

by

Kwaku Owusu-Tieku

August 2001

Dr. Donald Sanderson, Committee Chair

Dr. Martin Barrett, Committee Member

Dr. Phillip Pfeiffer, Committee Member

Keywords: Database, AP Modularization, STEP/EXPRESS, Software Engineering Metrics

ABSTRACT

Using Software Engineering Metrics In
AP Modularization
by
Kwaku Owusu-Tieku

Significant amount of work has been done in software engineering in terms of reuse. With the use of object-orientation and design patterns that support the development of reusable modules, it appears that the development and reuse of software modules in creating new systems is becoming more and more common. The software engineering world, however, has taken reuse more seriously than database; more research and improvement in reuse has been made in software engineering than in database. This paper investigates how software engineering metrics can be applied in the development of reusable database modules. This research provides a model for predicting the reusability of EXPRESS modules. It establishes a relationship between coupling and reusability of EXPRESS modules and provides a set of metrics that may be used in the proposed model for measuring coupling in EXPRESS modules.

ACKNOWLEDGEMENTS

I would like to thank my parents for their love and support and for their dedication to higher education. Without them, this would not have been possible. I would also like to thank Dr. Donald Sanderson, my thesis advisor, for his supervision and patience, Dr. Barrett and Dr. Pfeiffer, for advising and critique, Mr. David Price, Project Lead for AP Modularization effort, for helping me find literature on AP Modularization and sample EXPRESS modules. Your contributions are greatly appreciated. I would also like to thank all the students who participated in the survey. I could not have done this without them.

CONTENTS

LIST OF FIGURES

Figure Page

DEFINITION OF TERMS

Attribute. The term attribute is used in software and database modeling to mean characteristic of an object. Example: a name is an attribute of a person.

Class. A class is a model of a real world concept or object particular to object-oriented programming. A class specifies the prototype for a set of objects that share common characteristics and functionality. [http://www.instantweb.com/~foldoc/]. A class contains methods, which specify the functionality of the object, and attributes, which specify the state or characteristics of the object.

Database module. A database module is a data specification that models one or more related concepts. The term database module can be used to refer to a single entity or a set of entities in a schema that collectively describe an object or a concept.

Entity. An entity is a model of a real world object or concept particular to a type of database. In a database, a declaration of an entity introduces a new object into a data model and gives the characteristics (attributes) of that concept. The use of an entity in database is analogous to the use of a class in object-oriented programming.

Method. A method is an element of a class that specifies one aspect of the class's behavior. In procedural programming, a method is referred to as a function.

Object-oriented programming (OOP). Object-oriented programming is a school of thought that emphasizes the use of objects in programming. Solving a problem in OOP involves identifying what objects collaborate in carrying out the task and the responsibility of each object involved.

Product Data. The term product data here is used to refer to all information created or used by computer-aided design (CAD), computer-aided engineering (CAE), computer-aided manufacturing (CAM), and managed in a computer system [10]

Software Module. The term module refers to an independent piece of code that has a specific functionality. A software module can be as small as a single function. In the broadest sense, it can also refer to a class or a set of classes that collaborate to perform common task.

CHAPTER 1

INTRODUCTION

Statement Of The Problem

Reuse is the application of existing solutions to new problems. Reuse can reduce the time spent in creating solutions by avoiding duplicated efforts. In software engineering the concept of reuse has been explored and has been reported to be very beneficial. Frakes, for example, notes that "using reusable software generally results in higher overall productivity" [11]. According to Poulin et al. "the financial benefit attributable to reuse during the development phase is 80 percent of the cost of developing new code" [19]. The benefits are not only realized in productivity but also in quality; software developed using existing components can be more reliable than those developed from scratch because the reused components are usually well tested and have been used in several developments.

However, the reusable components must exist before they can be reused. The absence of formal reuse practices is, therefore, often not a result of unwillingness to practice reuse; rather the problem arises from lack of reusable objects. In both software and database, developers have produced large quantities of logic that cannot be reused due to its lack of structure and over-specificity. A partial solution to the problem of reuse, therefore, lies in the answer to the following question: What features make modules reusable, and how can one achieve such features in database design models? This research is an attempt to answer the above question.

Motivation

The research presented in this paper is motivated by the gains in productivity in software development due to reuse. While reuse has resulted in increased productivity and reliability in software [11], the concept and practice of reuse is still unexplored in database module design. One area in database design where reuse has recently received some attention is in the development of EXPRESS database modules known as Application Protocols or simply APs. EXPRESS is the data modeling language used in STEP. Application protocols are EXPRESS modules that form the unit of information exchange in STEP (See Chapter 3). Current

EXPRESS modules are huge, monolithic, and tailored to specific applications. In a process known as AP Modularization, developers are making efforts to design modules that are smaller, independent, and hopefully reusable. However, there are no guidelines as to what determines if a module is a good candidate for reuse or as to how a reusable module should be designed. Because object-oriented software modules bear a close resemblance to EXPRESS database modules, it is assumed that if metrics and guidelines similar to those used to develop reusable object-oriented software, are applied to the design of database modules, the gains in productivity seen in software may also be realized in database development.

Objectives

AP Modularization aims to achieve reusability through smaller modules designed to address single or closely related concepts. In software, the reusability of a module is determined by several factors, including coupling and complexity [24][18]. It is believed that in database several factors will also determine whether or not a module is reusable. The primary objective in this research is to determine whether or not coupling has effects on reuse of database modules.

Hypotheses

This research sets to establish whether or not coupling influences database module reuse. In statistics, a single hypothesis is usually expressed as two alternative hypotheses. The first part of a hypothesis is called the null hypothesis denoted by H_0. The second part of the hypothesis is the actual hypothesis (H_1) that is expected to be proven true. The proposed hypotheses are expressed below:

H_0: The time required to use an existing EXPRESS module does not increase significantly as coupling between the modules increases.

H_1: The time required to use an existing EXPRESS module increases significantly as coupling between the modules increases.

Thesis Outline And Approach

The outline of this thesis is as follows. Chapter 2 presents a study of reuse and its benefits in software engineering. Specifically, the features that promote reusability of software modules and the metrics for evaluating these features were identified. Selected metrics were chosen to serve as a basis from which specific metrics were derived for evaluating EXPRESS database modules. In addition, a description of proposed measurement model to be used in this study is presented.

Chapter 3 presents a study of the EXPRESS modeling language and AP Modularization and its goals, which form the basis for this study. In Chapter 4, the candidate metrics to be applied to a sample AP are identified and described. A detailed description of how the metrics were applied to the sample schema is also provided in this chapter. Chapter 5 presents the analysis of the results from applying the metrics to the sample schema. After applying the candidate metrics to sample schema, a survey was conducted to collect information about the difficulty in the use of EXPRESS modules. The survey asked participants to use existing EXPRESS schema items from the sample schema to which the candidate metrics have been applied. The intent was to record the amount of time taken to understand the selected modules. The analysis of this survey is given in this chapter. Chapter 5 presents both the findings from applying the metrics and the survey conducted and its results. Chapter 6 provides the final conclusion.

CHAPTER 2
REVIEW OF RELATED LITERATURE

Introduction

The background research in this paper involves two separate areas: software reuse and metrics and database schema design using EXPRESS. The first part of the research is devoted to reuse and software engineering metrics. The second part of the research focuses on STEP, an ISO standard of which EXPRESS is a part, including major features provided by EXPRESS for developing database modules.

Reuse And Reusability

Reusability: A Definition

Software reuse is the use of existing software components to construct new systems [20]. Reusing existing parts or components is a standard part of software engineering and human problem solving in general. However, reuse in software development is more effective if practiced formally [11]. Formal reuse implies that reuse must be viewed as a goal to strive for, not just a result that happens by chance. Before reuse can take place, the reusable components must exist in some form, and designers must be aware of their existence and the functionalities they provide. If formal reuse is part of an organization's overall development goals, then the software construction process is different; not only are developers tasked to find and use existing artifacts, they also have to assure that the final product can also be reused in future development. The task of storing and searching for reusable components can be streamlined using a populated repository of components that have been tested and proven reliable. In software engineering, such repositories exist in the form of user interface toolkits, frameworks, and libraries. In order to discuss the issues associated with the design of reusable modules, one must first understand the different kinds of reuse that exist.

Types Of Reuse

Software engineering literature lists many different kinds of reuse, but one of the most comprehensive lists is the one provided by Prieto-Diaz [20].

By Substance. Reuse by substance is categorized further into three sub-categories. Idea reuse involves reusing some existing idea that has been used to solve some recurring problem. Artifact reuse is the reusing of old components. Finally, procedural reuse is the reuse of exiting algorithms.

By Scope. Reuse by scope can either be vertical or horizontal. In vertical reuse, existing components are used to construct new applications within the same domain. In horizontal reuse, the components are used outside the domain for which they were originally designed. From design point of view, it may be easier to construct reusable components for vertical reuse than for horizontal reuse. Designing modules for horizontal reuse is complicated by needs to anticipate a larger scope and design the components in the most generic form to allow inter-domain application development.

By Mode. Reuse by mode entails the approach by which an organization conducts reuse. An organization may conduct reuse with a formal approach or in an ad-hoc manner. The state of practice in reuse in many software engineering organizations is characterized by an ad-hoc approach [20].

By Technique. Reuse can also be characterized by how the new system is actually built. A new system may be constructed by putting together existing components (compositional reuse), or by using high-level specifications and application and code generators to produce a new system (generative reuse).

By Intention. In reuse, whatever artifact is reused, it may be used as-is, or it may be modified or extended to provide additional functionality. The reuse of components without any modification is termed blackbox reuse. Whitebox reuse is when the component is modified

before use. According to Prieto-Diaz, whitebox reuse is prevalent in the current state of practice [20].

By Product. Reuse by product looks at what kind of artifact is reused. There are various products developed during the different phases of the software development. Although most of these products are developed without reuse in mind, they often become useful in new projects. These products include system architecture, high-level specification, design, objects, source code, and text.

Both software and database designers must be aware of the different facets of reuse. They should also keep the following in mind when designing reusable modules:

- Reusable components should be designed with the intent for reuse [1,2,9,27]. The major reason why the state of practice in software reusability is characterized by ad-hoc and whitebox reuse is that most software components are not designed for reuse. Existing software is not well documented; it is usually designed with restrictions that are specific to the current application. These factors limit the reuse of modules in other applications.

- Reusable components should be tested or certified [20]. The testing of modules assures quality and reliability. However, the size of the library and the complexity of the software complicate the task of testing a large library of modules.

- Reusable components should be classified and collected into accessible libraries [7,11]. Reuse cannot take place if the components are not accessible. In software organizations, reuse can be a very difficult task if components are not grouped together into some organized form.

- Reusable components should be accompanied by documented interfaces [11]. Developers often spend large amounts of time trying to find out what functionality is provided by frameworks and how to use them. The task of selecting and using components can be further complicated by the lack of documentation describing what components do.

What Is A Reusable Software Module?

Although software reuse is still practiced in an ad-hoc manner, improvements continue to be made in this field especially in the area of graphical user interface design. Frameworks such as the Microsoft Foundation Classes (MFC) and the Java Foundation Classes (JFC) simplify some complex tasks by providing generic solutions that can easily be applied to similar problems in the creation of graphical user interface applications. A reusable software module can be thought of as a unit of code or data specification that provides a specific functionality or semantic. The ideal features of such a module include functional independence, extensibility, and reliability. Functional independence is concerned with modules that perform single tasks. Extensibility is the ability to modify a module so that it performs new or additional tasks. Reliability is concerned with modules that produce the same results accurately and consistently. If creating software from reusable components is difficult, designing the reusable modules is even more difficult. For both designers and users of reusable modules, some of the questions that need to be asked include the following: What are the indicators of reusable modules? What criteria can be used to evaluate modules for reusability?

Designing Components For Reusability

The creation of reusable modules and the identification of such modules by developers is part of what makes reuse a difficult task. A design activity is a recursive decomposition of larger components or modules into desired level of granularity and functionality [17]. The art of decomposing larger components to achieve reusability requires an identification of modules that could be potential sources of features that may hinder reuse. The task of identifying error-prone modules requires that the factors that prohibit reuse be known so that both qualitative and quantitative guidelines or metrics can be developed for evaluating the modules. When such guidelines and metrics are devised, they may be used to pinpoint areas that need rework in the design, but first the indicators of reusability must be identified.

In order to devise a measurement model or qualitative guidelines for evaluating reusable components, the factors that are known to influence reuse must be identified. In software engineering, certain factors are known to influence reuse. These factors include coupling, cohesion, complexity, and modularity.

Factors Influencing Software Reuse

Coupling. Coupling is a measure of interconnection among modules [18]. In software design, the goal is to achieve low coupling among components. Low coupling will result in a system with independent components that are easy to understand, easy to maintain, easy to test, and possibly more reusable than highly coupled modules.

Cohesion. Cohesion is concerned with individual components having singleness of purpose [18]. In software, high cohesion is sought because high cohesion promotes modularity, which makes testing and maintenance less difficult [2].

Complexity. Complexity can be viewed in different ways. Algorithmic complexity is a measure of an individual algorithm's intricacy. Structural complexity is a measure of the system's interrelatedness: for example nesting, interdependence of objects, or inheritance. [9].

Modularity. Modularity in software is the division of large components into smaller manageable units each addressing a smaller part of the problem to be solved. Modularity reduces the complexity of a large program by breaking the problem into manageable units [18].

Metrics have been developed in software engineering to quantitatively measure these factors and such metrics have been used to assess software modules for reusability. In this research, the focus is whether or not coupling affects database module reuse.

Reusability Metrics And Models

The Factor, Criteria, Measurement (FCM) Model

In software engineering, several measures have been used to evaluate software quality. At minimum, for a component to be considered for reuse, it must be of good quality. Measuring quality quantitatively is not a simple task. As stated by Fenton et al., "quality is multi-dimensional; it does not reflect a single aspect of a particular product " [9]. Many software metrics text and papers [9,12] give models for measuring software quality. One of these models, proposed by Fenton and colleagues [8], define factor, criteria, and metric (FCM) for each measurement. FCM is a tree-like structure where the top level lists the factors—items that are

known to be the major indicators in the evaluation of the attribute in question. For instance, in evaluating quality, one may look at usability, testability, and portability as factors giving indication of the quality of a product. The second level in FCM consists of a list of criteria for each factor. These lower level items are easy to understand and measure. The last level comprises of the actual metrics that define the specific measurements for each criterion. For instance the criteria comment ratio may be defined as a criteria for evaluating understandability. For this criterion, specific metrics can include counting the number of comments lines per source line and counting the number of comments lines per components.

Proposed Measurement Model

Database Module Reuse — A Definition

In this research, a measurement model based on McCall's FCM model will be used for evaluating and predicting the reusability of EXPRESS database modules. The proposed measurement model is shown in Figure 1. In the proposed model, reusability is the final goal. The major factor chosen as the indicator of reusability with respect to database modules is understandability. Understandability is a qualitative attribute and, hence, is difficult to measure directly. Coupling is used as an indirect measure of understandability. The major assumption here is that in order to reuse a module, one needs to see the definition of that module in an attempt to understand it. Understanding the module can be complicated if that module is coupled with many other modules. Therefore, the degree of coupling in a module can be an indication of the effort required to understand the module, which can affect reuse. Specific metrics are hence chosen to measure the features in EXPRESS database modules that introduce coupling. The following sections will provide a definition of reusability with respect to database modules as used in this research.

One of the goals of this research is to be able to recommend metrics that can be used to evaluate and predict the reusability of EXPRESS database modules. The definition of reusability, as used in the proposed measurement model, is based on the reuse model of AP Modularization described in Chapter 4. A database module is said to be reusable if it can be

a) used as part of another application or as part of a larger module without any modification to it, or

b) modified to add extra functionality (extension) before using it in another application or as part of a larger module, or

c) modified to restrict its domain or scope (specialization) before using it as part of another application or as part of a larger module.

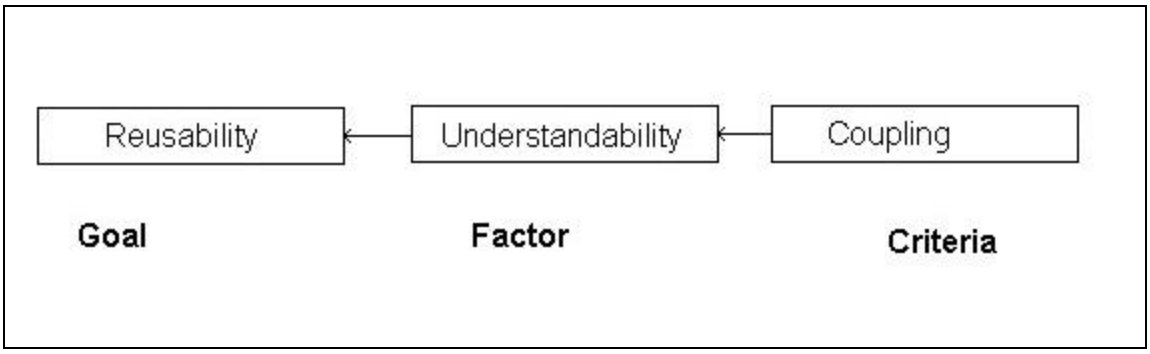

Fig 1: Proposed Measurement Model

Proposed Model

Factor: Understandability. In this research, understandability is defined as the ability to comprehend a module (in terms of time taken to understand it) given the minimum or no internal documentation. The level of difficulty or the amount of time required to understand a module is important because developers using a module need to understand both syntactic and semantic aspects of a module to be able to make changes to the module. The level of understandability of a module is related to the coupling within the module. The more coupled a module with other modules, the harder it is to comprehend it.

Criteria: Coupling. Coupling is chosen as a criteria in determining the reusability in the proposed model not only because it is often quoted as one of the determining factors in software quality [18] but also because a number of coupling measures are mentioned in object-oriented metrics [2][5][17].

Survey Of Software Engineering Metrics

Introduction

Recent research in software design metrics has emphasized complexity, especially, design complexity, and reusability. Some of the classical complexity measures described by Fenton et al. [9], including Cyclomatic Complexity, have been used in evaluating quality in procedural software. Another trend has been to focus on object-oriented metrics to capture features of object oriented software. Factors addressed include complexity, reusability [6,22], and maintainability [3]. Emphasis has shifted from code (algorithmic) complexities to design complexities capturing features such as complexity of inheritance hierarchies [17,24], coupling and cohesion [2,17], and interface complexity [5,15].

Some Software Design Metrics

This section lists some software engineering metrics that have been used to measure software quality. The purpose of this list is to provide candidate metrics that can be used or adapted for use in the proposed measurement model.

1. Source Lines Of Code (SLOC) [5]. SLOC is the simplest of the traditional code-level metrics that use program size to determine program effort and complexity. Various forms of this metric exist depending on what is deemed to be important. Because software today can be generated by reuse of existing products and also by automatic code generators, this metric has become less important.

2. McCabe's Cylcomatic Complexity (CyC) [5]. First proposed by McCabe in 1976, this metric uses directed graphs to capture the algorithmic complexity of a module. McCabe proposed that the higher the value for this metric, the more complex a module. The unit of measurement is a module or function.

3. Class Method Complexity (CMC) [5,15]. Originally proposed by Chidamber and Kermerer (C&K) as WMC (Weighted Method Per Class), Li redefined two metrics, the CMC (Class Method Complexity) and NLM (Number of Local Methods), to capture what the WMC was designed to measure. CMC is a sum of the weighted values for method complexity. The

weighted complexity can be calculated using for, instance, McCabe's Cylcomatic complexity. The rational for this metric is that the more methods in a class and the higher the values for their weighting factors, the more complex the class, which makes it more difficult to use. The unit of measurement is class method.

4. Number of Local Methods [15]. NLM is also one of the object-oriented metrics proposed by Li. NLM measures the total number of local methods per class. The unit of measurement is class method.

5. Average Method Complexity (AMC) [5]. This metric is a modified version of Chidamber and Kermerer's WMC. It considers the average method complexity as a good indicator of overall complexity rather than the sum.

6. Number Of Variables (NAV) [22]. Mentioned by Reyes and Carver, this metric measures the total number of variable in a class. The unit of measurement is class variable.

7. Depth Of Inheritance (DIT)[5]. This metric is one of the metrics from C&K suite. It measures how deep a class is in an inheritance hierarchy. The unit of measurement is class. The viewpoint is based on the fact that the deeper a class is in an inheritance hierarchy, the more complex it becomes, because many classes higher in the hierarchy can affect it.

8. Number Of Ancestors (NAA) [5,15]. Li tries to capture the effect of inheritance hierarchies on classes by defining the NAA metric. His metric is more specific than the C&K's DIT because it captures exactly which classes can affect another class by inheritance. It is a count of all classes that a class inherits from.

9. Number Of Descendants (NOD) [15]. This metric is also proposed by Li. The metric measures the number of classes that inherit from a specific class.

10. Response For Class (RFC) [5]. Also one of the metrics from the C&K metrics suite, this metric measures the potential communication between classes. It is a count of all methods in a class including other methods called by these methods.

11. Data Abstraction Coupling (DAC) [15]. Data Abstraction Coupling is referred to as coupling through abstract data type and is defined as total number of classes that are used as abstract data types in the declaration of a data attribute.

12. Specialization Index (SI) [12]. According to Gillibrand et al., this metric gives an indication of how well a subclass fits a hierarchy in which it is placed. For instance, if a subclass makes less use of inherited methods and attributes and instead adds several new ones or overrides inherited methods, then it may suggest that either the parent class does not correctly model the real concept or the subclass does not belong in that hierarchy. The SI is defined as follows:

$$SI = \frac{\text{number of overridden methods} * \text{class hierarchy nesting level}}{\text{Total number of methods}}$$

13. Inheritance Level Technique (ILT) [24]. Mentioned by Shih et al., this metric attempts to capture the complexity of inheritance hierarchies. ILT models an inheritance hierarchy using a directed graph where every node represents a class and edges represent parent-child relationships. The metric is based on a single unit called unit repeated inheritance (URI). The URI is defined as a directed acyclic graph that has the same number of edges as node [24]. The complexity of an inheritance hierarchy can be indicated by value of ILT metric, which is the summation of URIs at all levels of the hierarchy.

CHAPTER 3
STEP/EXPRESS AND AP MODULARIZATION

STEP

Introduction

This section introduces STEP, EXPRESS, the STEP Application Protocol, and their relevance to this study. Briefly, STEP is an international standard for information interchange [25]. EXPRESS is a data modeling language provided as part of the STEP standard for describing the information to be exchanged [10]. The Application Protocol (AP) is a single unit of information (EXPRESS information model) that is exchanged using STEP [10]. The major motivation behind this study arises from the need for the design of modular APs. Further details about STEP, EXPRESS, and APs are provided in the following sections. Before the STEP standard is discussed, a brief discussion of the problem of data exchange is presented.

The Data Exchange Problem

In the manufacturing and engineering industries, there has always been a need to share product data. The term product data is used to refer to all information about a company's products and processes that are created and managed in a computer system [28]. The product data describes all information about a product through its life-cycle. Often a company is spread across different geographical sites, and data need to be exchanged between those sites or sometimes between a supplier of a product and a user of that product. In the past, lack of data formats for exchange has resulted in an inability to share data, or in loss of information during exchange. Information was lost because different parties often implemented different exchange standards. Even in cases where the same standard was implemented, different subsets of the standard were implemented in different software applications. Hence translation from one software application to another resulted in only a part of the information being translated. Some earlier exchange standards include IGES, DXF, and SET [28]. All these standards attempted to provide a solution to the data exchange problem by providing a single standard within some industries. Each standard, however, focused on a limited scope and failed to provide a

comprehensive solution to the data exchange problem [21]. For instance, IGES (Initial Graphics Exchange Standard), developed in the 80s [21], focused only on CAD products. SET (Standard d'Exchange et de Transfer) was the French response to the data exchange problem, and again its scope was limited to CAD data.

What Is STEP?

STEP is an acronym for Standard for the Exchange of Product data. STEP is an ISO standard with designated name ISO 10303: "industrial automation systems — product data representation and exchange" [10]. The major objective of STEP is to provide a solution to the data exchange problem faced by CAD/CAM and the manufacturing industries by specifying a neutral format for exchanging data. STEP provides a standard way for describing product data, with mechanisms for implementations and testing for conformance. The standard is comprised of series of parts that are published separately. Each part is numbered and is designed to address a separate aspect of the standard. The initial parts of STEP were accepted and published as an international standard in 1994, but the standard continues to evolve.

Key objectives for STEP as given by Fowler [10] include the following:

- Provide a single international standard that covers all aspects of CAD/CAM data exchange.
- Provide a standard way of describing product data throughout its lifecycle, independent of any computer system.
- Separate the description of data from its implementation to make the standard suitable for neutral exchange. Separating the description of data from its implementation also will allow the standard to act as a "basis for shared databases and for long-term archiving." [10].

STEP Architecture And Components

The STEP standard is organized as a multi-part standard that supports the decoupling of data description from implementation and testing. The complete structure of STEP is shown in Figure 2. The core architecture of STEP mirrors the three layers in the ANSI/SPARC model upon which STEP was modeled [10].

The ANSI/SPARC three-layer architecture emphasizes the identification and separation of three key items in database design. The highest level of the architecture is the application

layer, which consists of users' views of the systems. The next layer is the conceptual or logical layer. This layer provides an application-independent data models that can be implemented by different users at the application level. The lowest level of the architecture is the physical layer. This layer consists of data structures, which implement the conceptual layer.

The core architecture of STEP can be compared to the ANSI/SPARC three-level architecture. The Application Protocols represent the specific application views in STEP. The APs correspond to the application level in ANSI/SPARC three-layer architecture. The Integrated Generic Resources, which include the new modularized Application Modules (AMs), Integrated Application Resources (IAR), Integrated Generic Resources (IGR), and Application Interpreted Constructs (AIC), correspond to the logical layer in the ANSI/SPARC three-level architecture. Finally, the implementation methods, which provide standard mechanisms for encoding data for exchange and methods for accessing such data, correspond to the physical layer in the ANSI/SPARC three-level architecture.

The contents of STEP can also be divided into two major categories: infrastructure and information models [14]. STEP's infrastructure consists of Description Methods (Parts 11-19, including the EXPRESS language), Implementation Methods (Parts 20-29), and Conformance Testing methods (Parts 30-39). The Implementation Methods describe ways for physically encoding data for exchange and for providing access to such data in software applications. The Conformance Testing Methods describe procedures for testing STEP implementations for conformance to the standard.

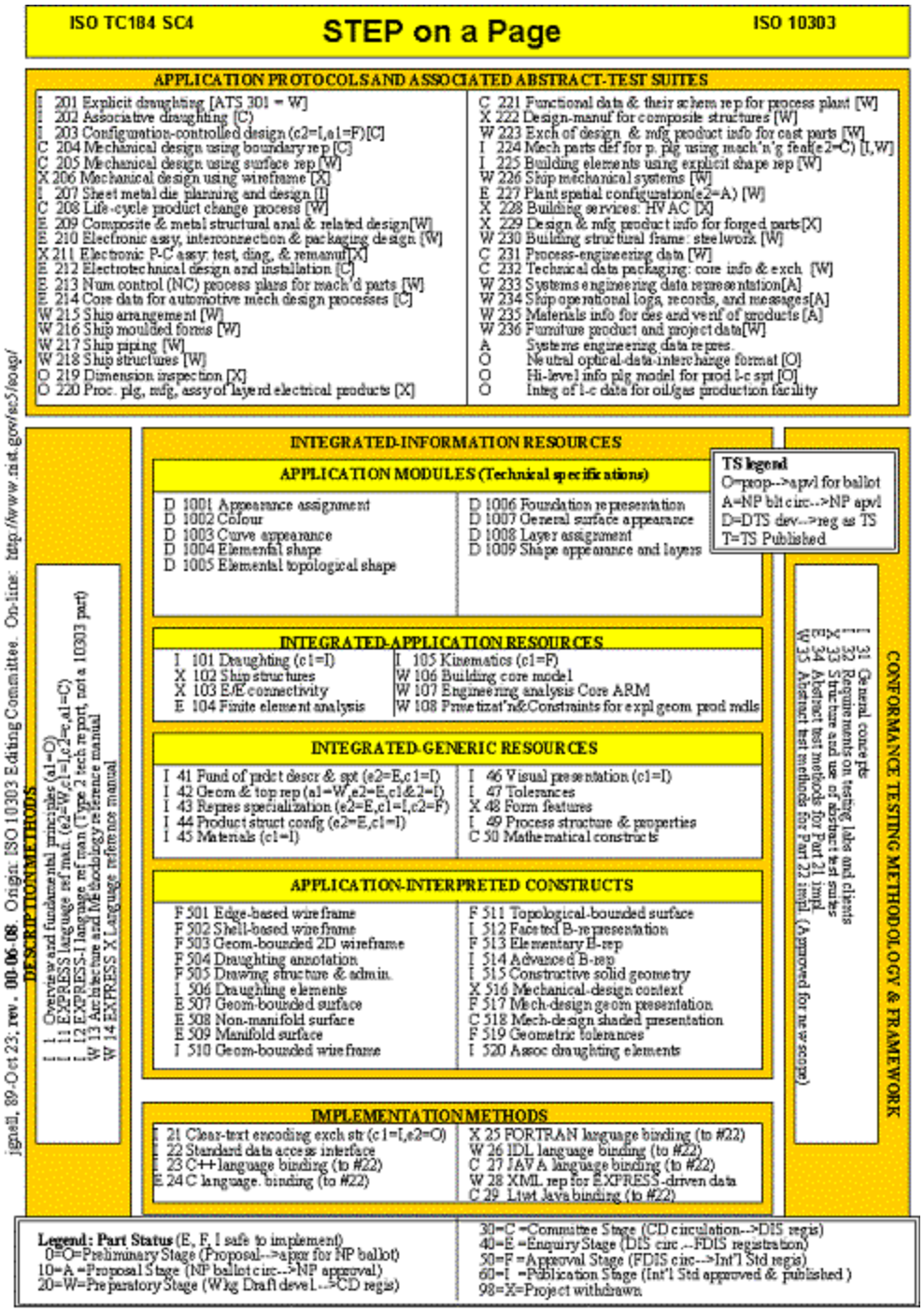

Fig. 2: Structure Of The STEP Standard. From http://www.nist.gov.stepdocs/htm

The information models consist of Application Products (APs) and Integrated Resources for building APs. More detail about AP development is provided in the following sections. The information models constitute a "set of entities chosen for a specific product, process or

industry" [13]. The AP is built from using two sets of resources: Application Resource Model (ARM), and Generic Resources.

The Application Protocol (AP)

The bulk of the STEP standard is made up of Application Protocols (APs). An AP is the final product in STEP development; it is a specialized set of entities with specific business rules that constrain and define the collection of information that forms the basic unit of exchange in STEP. Technically, the AP is made up of the Application Activity Model (AAM), the Application Reference Model (ARM), and the Application Interpreted Model (AIM). The Application Activity Model describes the activities of the product's lifecycle [14]. It includes a high-level description of input, output, and processes for in a particular domain. The Application Reference Model describes product information needed in the Activity Model; it resembles a Software Requirements Specification (SRS) in content. The Application Interpreted Model is an EXPRESS schema that defines a formal information model, which captures everything specified in the ARM; it specifies all the information that is to be exchanged. A Mapping Table is used to translate contents of the ARM into generic constructs defined in the Integrated Resources to produce the AIM. The AP also includes Conformance Classes, which specify the minimum subset of the AIM that must be implemented in order to conform to the STEP standard.

Resources For AP Development

Another part of the STEP standard is called the Integrated Resources (IR). The IR sub-layer ensures consistency in APs across different applications by providing standard data specifications for developing new APs. IR modules can be regarded as building blocks of STEP. Currently there are three types of resources in STEP. The first set of resources is called the Generic Resources (GRs). The Generic Resources provide the most generic data specifications in STEP information models and can be used across all parts of STEP AP development. Another section of the Integrated Resources is the Application Resources (ARs). Parts in this section are numbered in the 100s and contain entities that are more application-specific than those in the Generic Resources. The last set of the Integrated Resources modules is the Application Interpreted Constructs (AIC). These consist of data specifications that have identical semantics in two or more applications. For instance, a data specification for a date usually retains the same

meaning even in different applications and may in fact be used in different applications. The AICs are numbered in the 500s.

The AP Development Process

The development of an Application Protocol (AP) in STEP is initiated by industry needs or by new technologies and techniques [10]. This implies that, in STEP, APs are not developed without a prior need. This requirement assures that every AP or data in an industry application that conforms to the STEP standard can be traced to the reason of its existence [10]. The purpose of an AP is to provide a standard description of an industry application including the scope and purpose of such application, the activities involved, input and output data, and methods for exchanging such information. To ensure consistency, the APs are developed by selecting and reusing standard data specifications or constructs from the Integrated Resources. The term data specification refers to descriptions that provide facts about an object [10].

The first task in developing an AP is to gather the industry needs, usually from domain experts. The next step is to develop the APs Application Activity Model (AAM) and the Application Reference Model (ARM). Next, a Mapping Table is provided to relate the contents of the ARM (mostly business terms and descriptions) to standard data specifications provided in the Integrated Resources. This Mapping table is the basis for the Application Interpreted Model (AIM). The AIM is the final product in AP development, although it is not the AP itself. The Scope of Application Protocol, Constraints, and Conformance Classes (CC) are added to the AIM as a final step in the AP development. The Conformance Classes define the minimal subset of the AIM that must be implemented for conformance to the STEP standard.

EXPRESS

Introduction

EXPRESS (ISO 10303-11) is the designated modeling language for STEP. EXPRESS constitutes a major part of the Description Methods, which are a fundamental part of the STEP standard. The role of EXPRESS in STEP is to define the syntax of the information models that describe data to be exchanged. EXPRESS is an object oriented modeling language. It contains

features that are very similar to those found in object-oriented languages like C++. The domain analysis and the extraction of entities in EXPRESS modeling resemble the activities done when modeling software using an object-oriented methodology. EXPRESS, however, also supports constructs for information modeling, including features for creating data models and specifying rules and constraints independent of implementation. The following is a brief summary of major features in EXPRESS language.

Features Of EXPRESS

Schema. A schema is the basic building block of EXPRESS models. A schema is a container for all declarations and definitions that appear in a model.

Data Types. Data types specify the domain for which instances can assume values. EXPRESS provides numerous data types, which can be used in various ways. Attributes and parameters defined in a schema must have underlying data types that define their domains.

```
SCHEMA test;
        TYPE … END_TYPE;

        ENTITY … END_ENTITY;
        ENTITY … END_ENTITY;
END_SCHEMA
```

Fig. 3: A Schema In EXPRESS

Simple Data Types. EXPRESS provides simple data types as the basis for defining user-defined types. They provide the domain for the atomic data that cannot be further subdivided. The simple data types include NUMBER, REAL, INTEGER, STRING, BOOLEAN, LOGICAL, and BINARY.

Aggregation Data Types. Aggregation data types, sometimes called collection data types, define a domain that consists of a collection of values of one simple data type. The size of these collections can be fixed or varying depending on optional constraint present in the type's declaration. The aggregation data types include ARRAY, LIST, BAG, and SET. An ARRAY is an indexed, unordered collection of elements. Whether the array can contain duplicates or not

can be specified at declaration using the UNIQUE keyword. A LIST is a totally ordered collection of elements. Lists may contain duplicates, unless explicitly prevented by the use of the UNIQUE keyword in the declaration. A BAG is a collection of unordered elements in which duplicates are allowed. A SET is unordered collection of elements in which duplicate instances are prohibited.

```
SCHEMA schoolInfo;

  ENTITY student;
     ID         : STRING;
  END_ENTITY;

  ENTITY book;
     title         : STRING;
  END_ENTITY;

  ENTITY book_shelf;
    books       : ARRAY[0:?] OF book;
  END_ENTITY;

  ENTITY bag_pack;
    books       : BAG[0:?] OF book;
  END_ENTITY;

  ENTITY organization;
    Members : LIST[1:?] OF student;
  END_ENTITY;

  ENTITY class;
    the_students: SET [1:?] OF students;
  END_ENTITY;

END_SCHEMA;
```

Fig. 4: Aggregation Types

Constructed Data Types. EXPRESS provides two types of constructed data types. They are ENUMERATION and SELECT. These types are part of what EXPRESS calls DEFINED data types and they are declared by the keyword TYPE.

a) ENUMERATION Type. The ENUMERATION data type defines an ordered list of names.

b) SELECT Type. SELECT defines a data type whose domain is a union of the domains of the types specified in the select list. It is used to define a set of values from which an instance of an attribute can assume one and only one of those values. The data type defined by SELECT is usually a generalization of the types specified in the select list. Specified in the select list must be constructed types that are visible within the scope of the schema. Figure 5 shows a declaration of ENUMERATION and SELECT data types.

```
SCHEMA test;

  TYPE employee = ENUMERATION OF
        (temporary,  permanent);
  END_TYPE;

  TYPE contractor = ENUMERATION OF
      (government, private);
  END_TYPE;

  TYPE agent = SELECT
      (employee, contractor);
  END_TYPE;

END_SCHEMA;
```

Fig. 5: Constructed Types

Named Data Types. In EXPRESS, data types are used in various ways. Some are used as underlying data types for attributes. Others are used for declaring formal parameters and return types for functions. The only types that can be declared in a formal specification (in a schema) are

NAMED data types. EXPRESS provides two kinds of named data types: the ENTITY data type and the DEFINED data type

a) Entity Types. An entity in EXPRESS describes a single concept like a class does in object-oriented programming. A declaration of an entity contains a list of attributes that describe that entity. An entity declaration may also include rules and function calls to constrain instances of attributes of that entity. Figure 6 shows an entity declaration in EXPRESS.

```
SCHEMA studentType;

  ENTITY student;
    ID    : STRING;
    Name :STRING;
    SSN   : STRING;
  END_ENTITY;

END_SCHEMA;
```

Fig. 6: Entity Declaration

b) Defined Types. A defined data type is declared by the use of the TYPE keyword. A defined data type allows a designer to define a new type from an existing type by adding constraints and assigning a new type identifier. ENUMERATION and SELECT types are also part of the DEFINED types. Figure 7 shows the use of a defined type that restricts the domain of simple type based on an INTEGER.

```
SCHEMA colorType;

  TYPE color_value = INTEGER
    WHERE (SELF >0) AND (SELF <=255)
  END_TYPE;

  ENTITY color;
    R: color_value;
    G: color_value;
    B: color_value;
  END_ENTITY;

END_SCHEMA;
```

Fig. 7: Example Use Of Defined Type

Derived Attributes. Databases, as a rule, contain some values that can be computed from other values, and do not need to be stored physically in the database. For instance, a person's age can be computed from the date of birth. EXPRESS provides a construct for defining a derived attribute. The designer must specify the data type for the derived attribute as well as the expression or a function call that computes the value. Figure 8 shows how a derived attribute may be defined in EXPRESS.

Rules. In EXPRESS, rules constrain values that attributes may assume. EXPRESS provides two mechanisms for specifying rules: local rules and global rules. Also known as domain rules, local rules are defined inside an entity or defined type to constrain the domain of the attributes in that entity or type. An example of a local rule is given in Figure 9. Global rules are defined at a global level in the schema (outside all entities) and are used to constrain a set of entities in the schema.

```
SCHEMA saleType;

  ENTITY the_sale;
    sale:      REAL;
    tax:       REAL;
    DERIVE
    tax_amt:  REAL:= sale * tax;
  END_ENTITY;

END_SCHEMA;
```

Fig. 8: Example Use Of Derived Attribute

Functions And Procedures. In EXPRESS, functions express algorithms that can manipulate their parameters and return values. Procedures are used merely to enforce some constraint; no value is returned. An example of a function definition in EXPRESS is shown in Figure 10.

```
SCHEMA dateType;

ENTITY Date;
   month : INTEGER;
   day    : INTEGER;
   year   : INTEGER;
   WHERE
      mm : month  <=12  AND month >0;
      dd : day    <=31  AND day >0;
      yy : year   >2000;
END_ENTITY; -- end Date

END_SCHEMA;
```

Fig. 9. Local Rules

Inverse Relationships. In EXPRESS, relationships between two entities can be represented by using the type of one entity as an attribute of another entity. Suppose, for example, that a student is a member of an organization.

The two entities student and organization are defined as in Figure 11. Usually only one part of the relationship is made explicit while the other part is implicit. For instance, Figure 11 explicitly shows the link from the student to the organization by the attribute member_of in entity student. The implicit relationship between the organization and

```
SCHEMA functionDef;

  FUNCTION average ( var1, var2:  NUMBER)
                                 : NUMBER
    LOCAL:
     sum       : NUMBER;
     avg : NUMBER;
    END_LOCAL;

    sum := var1 + var2;
    avg := sum/2;

    RETURN (avg);

  END_FUNCTION;

END_SCHEMA;
```

Fig. 10: A Simple EXPRESS Function

the students can also be made explicit by declaring an attribute in entity organization and using the INVERSE construct to indicate the reverse relationship.

```
SCHEMA studentOrg;

  ENTITY student;
    ID        : STRING;
    member_of : organization;
  END_ENTITY;

  ENTITY organization;
    name :     STRING;
    INVERSE
    members : SET[1:?] OF student FOR
member_of;
  END_ENTITY;

END_SCHEMA;
```

Fig. 11: An Inverse Relationship

Supertypes, Subtypes, And Inheritance. EXPRESS allows an inheritance hierarchy to be defined by the use of the SUPERTYPE and SUBTYPE keywords. The SUPERTYPE construct is used to define a supertype entity in an inheritance hierarchy. In a SUPERTYPE clause, one specifies all the entities that are subtypes of the supertype being declared. The SUBTYPE construct defines a subtype entity, i.e. an entity that inherits from specified set of supertype entities. The SUBTYPE clause must name all the entities that are supertypes to the defined subtype entity. Figure 12 shows the use of the SUPERTYPE and SUBTYPE constructs. EXPRESS also provides ways to restrict valid instances of entities in an inheritance hierarchy. For instance, to specify that a student entity can be undergraduate or a graduate but not both, the ONEOF construct can be used with the SUPERTYPE keyword (Figure 12). Similarly, the ANDOR constraint can be used to show that a student (graduate or undergraduate) can also be fulltime or part-time (Figure 13).

```
SCHEMA studentSchema1;

ENTITY student SUPERTYPE OF
       (ONEOF  (undergrad_student,
graduate_student) );

  ID  : STRING;
END_ENTITY;

ENTITY undergrad_student SUBTYPE OF
(student);
END_ENTITY;

ENTITY graduate_student SUBTYPE OF
(student);
  isGA  : BOOLEAN;
END_ENTITY;

END_SCHEMA;
```

Fig. 12: Inheritance with ONEOF constraint

Schema Interfacing. In Express, a schema is a container for entities, types, and rules. Often there is no single context in which all the elements in the schema may be used. Some definitions, however, may be used more appropriately in some

context than others. Schema interfacing allows for dedicated contexts to be composed from elements in other schemas. Schema reuse is achieved through the use of two EXPRESS constructs: USE and REFERENCE. These constructs import definitions from other schemas into new ones. Entities imported by the USE keyword become first-class elements in the new schema. These imported elements behave as if they were originally defined locally in that schema. Instances of these elements can independently exist in an information base defined using this schema. On the other hand, definitions imported by REFERENCE become second-class elements in the new schema. REFERENCED elements cannot have independent instances in an information base defined using that schema; any use of instances of items in the referenced schemas must reference instantiated items in the original schema.

```
SCHEMA studentSchema2;

  ENTITY student SUPERTYPE OF
   ((ONEOF
   (full_time, part_time))
   ANDOR
   (ONEOF (undergrad_student,
graduate_student)
   ));

    ID  : STRING;
  END_ENTITY;

  ENTITY full_time SUBTYPE OF (student);
  END_ENTITY;

  ENTITY part_time SUBTYPE OF (student);
  END_ENTITY;

  ENTITY undergrad_student SUBTYPE OF
    (student);
  END_ENTITY;

  ENTITY graduate_student SUBTYPE OF
(student);
    isGA       : BOOLEAN;
  END_ENTITY;

END_SCHEMA;
```

Fig. 13: Inheritance With ONEOF And ANDOR Constraints

Schema interfacing can be used to create schemas that are tailored to specific contexts by selecting only relevant entities. One technique for schema interfacing, subtype pruning, is a method for importing entities without their subtypes. A second form of schema interfacing, chaining, is the imports definitions into a schema indirectly, by including schemas that also import other definitions. Chaining is possible because items imported into other schemas with USE become local to that schema—hence, importable into other schemas. EXPRESS imposes no limit on how many times a type can be imported.

AP Modularization

Introduction

The Application Protocol (AP) is the basic unit for information exchange in STEP. The current state of practice has been that when a need arises for a new AP, development begins from

scratch. Like software modules, existing APs were not designed with reuse in mind; it is difficult to apply existing modules to new applications. Recent STEP meetings and workshops have discussed the possibility of creating modules that are generic and designed to allow further extension and reuse.

Goals Of AP Modularization

The goal of modularized Application Protocols, like in software, is to reduce development time and effort, which translates to a reduction in cost. The STEP AP initiative aims to create reusable AP modules by 1) separating business use from data specification, 2) separating conformance classes from data specifications, and 3) delaying the definition of scope and domain till a later stage (usually left for application developers). Modularization aims to allow what is known as AP interoperability, which refers to the ability to "reuse data created by implementation of one AP by an implementation of another AP" [16].

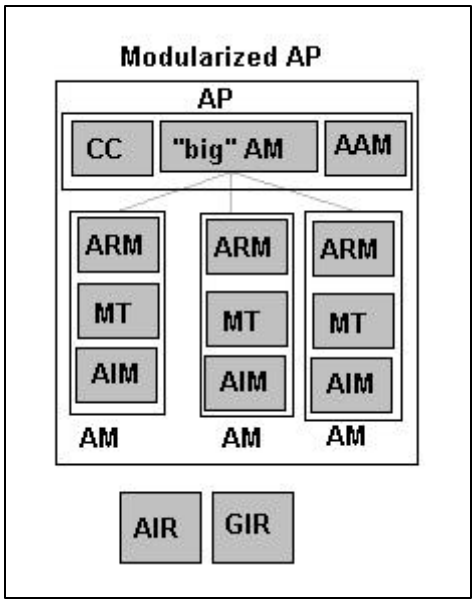

Figure 14: Structure Of A Modularized AP

Structure Of The Modularized AP

A modularized AP is made up .of Application Modules (AM). The Application Module is the basic reusable construct in the modularized AP. The AM is a data specification that contains the Application Reference Model (ARM), Mapping Table (MT), and a Module Interpreted Model (MIM) [16]. The Mapping Table shows how items in the ARM translate to generic constructs available in the Integrated Resources [21]. The Module Interpreted Model (MIM) refers to an AIM for a specific Application Module. Figure 13 shows the structure of the modularized AP. Unlike the non-modularized AP, the modularized AP does not contain the ARM, Mapping Table, and the AIM; it uses them by referencing Application Modules. Each AM contains the ARM, Mapping Table, and MIM.

In the modularized AP, the principal data specification (information model) is the Application Module. Each Application Module contains an ARM, AIM, and the Mapping Table.

The AMs are designed so that each AM defines an information model for one or more concepts. For the purpose of reuse, the AMs are designed with different levels of generality ranging from application specific to very generic. An AM can reference other AMs. In a modularized AP there is one application specific AM called the “big” AM. The “big” AM references other generic AMs, which in turn can reference other AMs. An AM may reference another AM for various reasons. For instance, an Application Module A may reference another Application Module B to define a specialization of a concept in Application Module A or to define a usage for an entity in Application Module A [25].

CHAPTER 4
METRICS APPLICATION AND SURVEY ANALYSIS

Selected Metrics And Measurement Units

Introduction

One of the goals of this thesis is to be able use SE metrics to evaluate the quality of EXPRESS database modules. One difficulty with the goal is that software and database are different domains with their own languages. However, software design and database design have common goals, like maintainability and reusability. Furthermore, the modeling language being studied, EXPRESS, provides features that are comparable to features found in modern object-oriented software design and implementations. These correspondences between EXPRESS and OO programming languages make it possible to apply some software engineering metrics to database modules with little or no modifications. If similar metrics that are used in software engineering can be applied to EXPRESS modules, then AP Modularization can make use of such metrics.

The nature of existing APs (EXPRESS modules) is the major cause behind AP Modularization, the basis for this study. Current APs are single, monolithic units that contain all required definitions, such as entities, types, functions, and procedures, in one single EXPRESS schema. The monolithic nature of the APs hinders AP reuse. Hence, the purpose of AP Modularization is to develop APs with smaller, reusable modules (in this case AMs). Before presenting the metrics to be used, and the measurable units in EXPRESS schemas, the goals of AP modularization in terms of reuse and how these goals relate to the proposed measurement model will be described.

The Reuse Model For AP Modularization

The process of developing smaller, independent, and reusable APs in STEP terminology is called AP Modularization. For the purpose of this research, a simple model, as shown in Figure 15, is used to show the goal of AP Modularization.

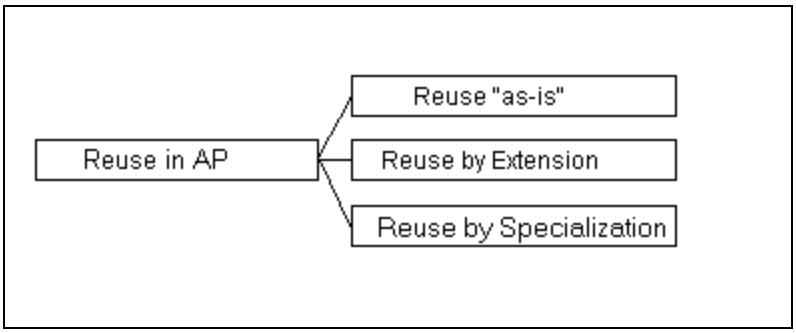

Fig. 15. Reuse Model For AP Modularization

The reuse model identified here shows three different of kinds of reuse in AP development. These types of reuse are explained below.

Reuse "as-is". A reusable component of an AP (i.e. AM) can be used without any change to it. This form of reuse is referred to as *Reuse "as-is"*. In this research, Reuse "as-is" is the type of AP reuse being investigated. The survey conducted sought to determine the effect of coupling on Reuse "as-is".

Reuse By Extension. A module can also be modified by extending it (adding new items to the data specification). Here, this kind of reuse is referred to as Reuse By Extension.

Reuse By Specialization. A module can be modified for the purpose of specialization (add scope or restriction to existing module). This kind of reuse is referred to as Reuse by Specialization. Figure 13 shows the reuse model from the AP modularization point of view.

Description of Proposed Candidate Metrics

This section lists and describes the actual metrics that will be used in the proposed measurement model. In addition, the reason for choosing each metric will be provided as well as how each metric fits in the overall goal of this research.

Data Abstraction Coupling (DAC). In EXPRESS, an attribute may have its type as one of the EXPRESS base types, often called primitive types: e.g. STRING, INTEGER, and NUMBER. An attribute may also have its type as a user-defined type. A user-defined type in this case may

be an Entity type, or an Enumeration type, or a Select type. DAC is an object-oriented metric proposed by Li [15]. This metric measures the use of classes as data types in attribute declarations. The measurement unit is a class. The viewpoint of DAC is that the use of other classes as types in the declaration of attributes introduces coupling between those classes. In this research, different versions of DAC will be used. The different forms of DAC that will be used in this research are listed below.

Data Abstraction Coupling from Entity Types (DAC_ENT). This metric will be used to determine the number of entities that have other entities as data types in their attribute declarations.

Data Abstraction Coupling from Enumeration Types (DAC_ENUM). This metric will be used to determine the number of entities that have Enumeration types as data types in their attribute declarations.

Data Abstraction Coupling from Select Types (DAC_SEL). This metric will be used to determine the number of entities that have Select types as data types in their attribute declarations.

Data Abstraction Coupling from Restricted (Defined) Types (DAC_DEF). This metric will be used to determine the number of entities that have Restricted types as data types in their attribute declarations.

Number of Supertypes (N_SUP). N_SUP is based on Li's Number of Ancestors (NAA) metric. Li's NAA is similar to C&K's DIT but NAA captures exactly which classes can affect a specific class in a hierarchy. In a complex inheritance hierarchy where a class may inherit from multiple parents, NAA is useful for tracing all the parents of any given class. The proposed metric, N_SUP, will be used to count the number of entities that a given entity inherits directly from. A high value for N_SUP may indicates that a class has a high risk of being affected by a change in many classes (supertypes).

Number of Subtypes (N_SUB). This metric is based on NOD (Number Of Descendants), an object-oriented metric proposed by Li [15]. NOD is a measure of the breath of an inheritance hierarchy. It is a count of the immediate children of a class. Like DIT, NOD assumes a view that the more children a class has, the more likely it is to reuse attributes and methods from the parent base class. However, a change in the base class affects the children. In this research, N_SUB is used (instead of NOD) to count the number of entities subclassing directly from another entity. Versions of N_SUB such as average N_SUB will also be used to evaluate a sample schema.

Depth Of Inheritance (DIT). Originally proposed by Chidamber and Kermerer [5], this metric measures the length of an inheritance tree from a node to the root (supertype). The viewpoint is based on the idea that the deeper a class is in an inheritance hierarchy, the greater the ability to reuse attributes and methods. However, deep inheritance hierarchies introduce complexity to classes because prior understanding of classes higher in the hierarchy is required in order to fully understand classes in the lower parts of the hierarchy. Another downside to deep inheritance hierarchies is that a change high in the hierarchy is more likely to affect classes lower in the hierarchy. The proposed measurement model is also expanded to show the metrics that are used in this research as shown in Figure 16.

Depth Of Data Abstraction Coupling (DAC_DEPTH). Attributes that use EXPRESS base types and Enumeration types as data types involve coupling with EXPRESS primitive types. This form of coupling is considered negligible in this research. However, attributes that use Entity types, Select types, and Restricted types as data types become coupled (in a form of physical dependency) with those types. For instance, if an attribute X uses an Entity type T as a data type in its declaration, then X becomes physical dependent on entity T (X is coupled to T). If T is also dependent on another entity Y, for instance, via inheritance relationship, then attribute X indirectly becomes coupled to Y (transitivity). This form of coupling is measured by finding the longest path to the last entity type in such a transitive dependency. The metric used for measuring this form of coupling is called DAC_DEPTH.

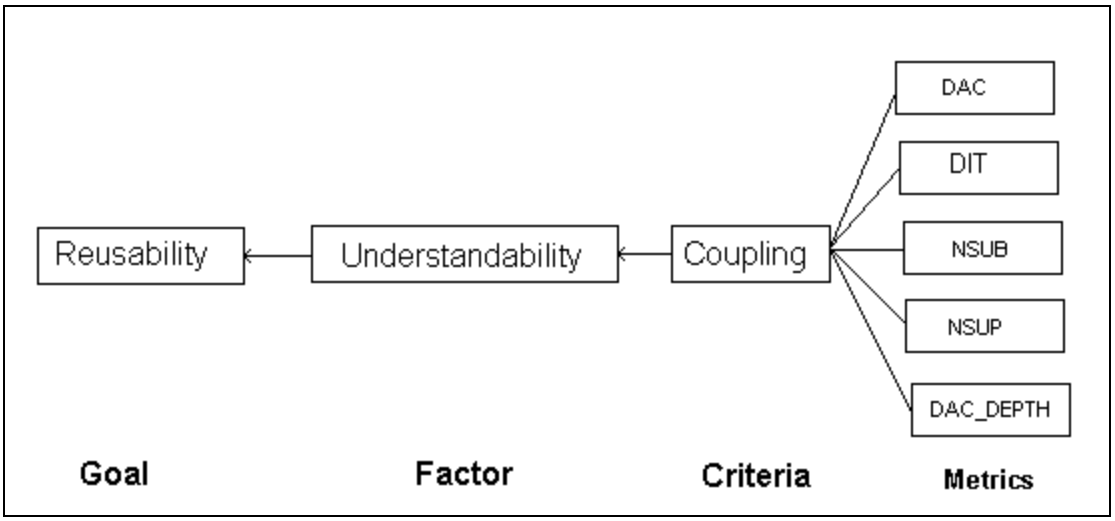

Fig. 16: Complete Measurement Model With Metrics

Applying The Metrics

The candidate metrics were applied to a subset of the EXPRESS module AP302 AIM (ISO 10303-203). This module was chosen for the following reasons. The version of the AP 203 used, besides being current (dated May 2000), also has a reasonable size. Although not too large, the AP 203 contains all the EXPRESS features that are being sought in this research. Due to its moderate size, survey participants (mostly students with basic EXPRESS skills) were more comfortable using it than it would have been with other APs that are published in the STEP standard.

Types Of Coupling Measured

There are several types of coupling found in EXPRESS modules. However, this research identified two most common forms of coupling:

- Coupling through data abstraction (DAC)
- Coupling through inheritance (C_INH)

The research focused on these two forms of coupling because in the current un-modularized APs, Data Abstraction and Inheritance are found in the majority of the type definitions in the schemas.

Coupling Through Data Abstraction (Data Abstraction Coupling - DAC). This type of coupling occurs when an attribute uses a user-defined type as its data type. In EXPRESS, a user-defined type can be created by:

1) Using an Entity type (DAC_ENT)

2) Using a Select type (DAC_SEL)

3) Using an Enumeration type (DAC_ENUM)

4) Using Restricted type (DAC_DEF)

Each of these user-defined types can, therefore, result in a data abstraction coupling. Figure 16-20 illustrates different forms of DAC and how they are measured.

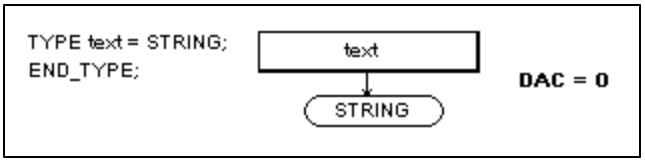

Fig. 17: DAC_DEF

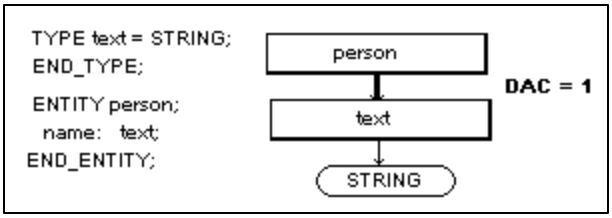

Fig. 18: DAC_DEF

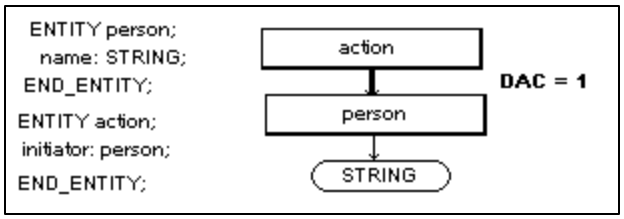

Fig. 19: DAC_ENT

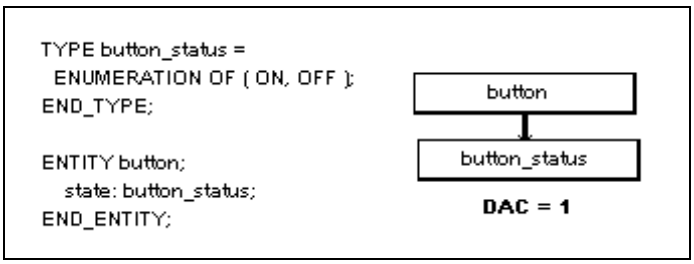

Fig. 20: DAC_ENUM

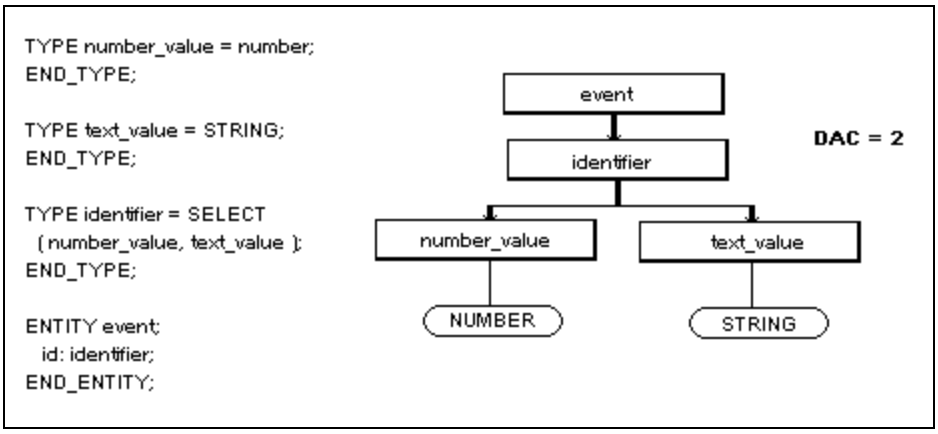

Fig. 21: DAC_SEL

In this research, data abstraction coupling due to the use of each of the four user-defined types is tested to see if they have any effect on reuse.

Coupling Through Inheritance (C_INH). This type of coupling occurs through inheritance. In an EXPRESS inheritance, the subtype entity is coupled to the supertype entity by referencing the supertype in its SUBTYPE clause. The supertype entity may also mention the names of all subtype entities in its SUPERTYPE clause, resulting in further coupling. Figure 20 illustrates the coupling though inheritance.

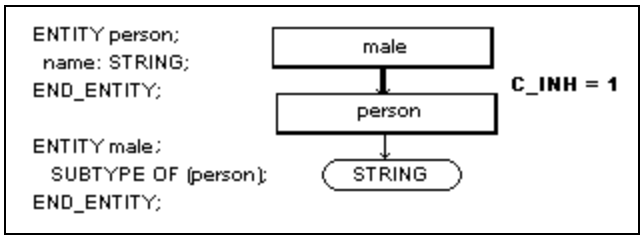

Fig. 22: Coupling Through Inheritance

CHAPTER 5
ANALYSIS

Analysis Of Sample AP

In this research, a sample AP (Appendix A) was selected for study to determine if coupling has any effects on the use of EXPRESS schema items.

The candidate metrics were applied to the sample EXPRESS schema to determine what features dominate the schema definitions. The sample AP was the single schema AP302 AIM (ISO 10303-203). The schema was modified to reduce the size and complexity so that students with minimum EXPRESS skills would be able to use it. The main features of the AP that were analyzed are use of inheritance and data abstraction. The following observations resulted from applying the candidate metrics.

Table 1: General Statistics About The Sample Schema

General Statistics : Type Composition		
	Number	Percentage
Number of Entities	151	71%
Number of Restricted types	28	13%
Number of Select types	31	14%
Number of Enum types	4	2%
Total	**433**	**100%**

Entity types composed 71% of all type definitions in the schema. There were 219 attribute types distributed in the 151 entities found the schema giving a very low average of 1.5 attributes per entity. Of the 151 entities, 43% had no attributes, while 59.6% had between one and three attributes; only 17% of the entities had four or more attributes. schema was six and only one entity had this number.

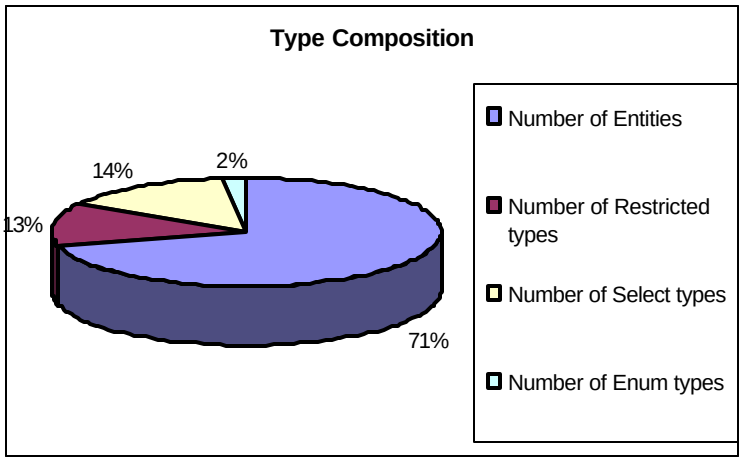

Fig. 23: Type Composition For The Sample Schema

The type composition also included 28 Restricted types which makes up about 13% of the total type composition. Of this 13%, none had more than one level of redefinition. In fact, over 80% of all the Restricted types were based directly on EXPRESS base types (zero level of redefinition).

Fourteen percent of the all the type definitions in the schema consist of Select types. Enumeration types make up 2% of the schema type definitions. In the analysis, it was found that only 1.8% of attributes used Enumeration types in their definitions, and 2.7% for Select types. Because Enumeration types do not reference any other types in their definitions, they do not add any form of coupling to the schema.

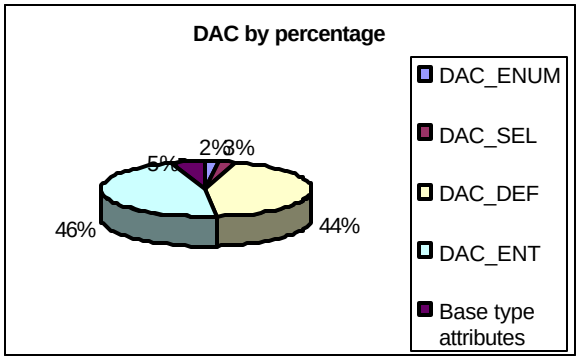

Fig. 24: Data Abstraction By Percentage

Attribute Types And Data Abstraction Coupling (DAC)

Data Abstraction Coupling (DAC) results when attributes use other user-defined types such as entity types as data types. In the sample AP, DAC_ENT was found to be the major form of coupling in the schema. DAC_ENT causes more physical dependency for entities than any other form of coupling in the schema. Of the 219 attributes found in the schema, 46% are involved in DAC_ENT (i.e. attributes that use Entity types as their data types), about 44% use Restricted types as data types. The remaining 10% use Enumeration types (DAC_ENUM), Select types (DAC_SEL), and EXPRESS base types as their data types (See Figure 24). In the schema analyzed, 44.5% of all the attributes had a DAC_DEPTH value of 2 or higher. The average value for DAC_DEPTH is 1.6, which shows that Data Abstraction Coupling in general is low for the schema.

Inheritance

Inheritance is another major cause of coupling in an EXPRESS schema. Entities become physically and logically dependent on each other through inheritance relationships. The schema that was analyzed made very little use of inheritance and, hence, coupling resulting from such relationships is minimal. Although out of the 151 entities in the schema, 62% were involved in an inheritance relationship, the average depth of inheritance (average DIT) is about one (1.1).

37.7% of all the entities were not involved in any inheritance relationship. Of the 111 entities involved in inheritance relationships, 19.2% were supertypes, and 7.3% were root supertypes. That means all the inheritance hierarchies in the schema are built on 7.3% of the entities. In terms of multiple inheritance, the majority of the entities in the schema (52%) have only one subtype, while only about 1 (1.3) % two or more supertypes. The value for DIT (the longest path from any supertype to a subtype) was found to be 3; the average DIT is 1.1. Table 2 shows the valued obtained from applying the metrics.

Table 2. Values Obtained From Applying The Metrics

Entity Analysis		
Avg No. of Attributes/Entity	1.5	
Avg DIT	1.1	
Deepest DIT	3	
DAC	**#**	**%**
DAC ENUM	4	1.8%
DAC SEL	6	2.7%
DAC DEF	96	43.8%
DAC ENT	101	46.1%
Base type attributes	12	5.5%
Max DAC_ENT_DEPTH	4	
Max DAC_SEL_DEPTH	3	
Max DAC_DEF_DEPTH	1	
Avg DAC_ENT_DEPTH	1.5	
Avg DAC_SEL_DEPTH	1.3	
Avg DAC_ENUM_DEPTH	0.0	
Avg DAC_DEF_DEPTH	1.0	
Inheritance	**#**	**%**
Total No. of Root Supertypes	11	7.3%
NSUP	29	19.2%
NSUB	82	54.3%
Total No. of Entities with inheritance	94	62.3%
Total No. of Entities without inheritance	40	37.7%
Max No. of Supertypes / Entity	5	
Max No. of Subtypes / Entity	15	
Avg No. of Subtypes / Supertype	3	
Number of Complex Entities	2	
Max No. of attributes / entity	12	

DAC_ENT_DEPTH		
Range	Frequency	%
0	7	6.9%
1	49	48.5%
2	27	26.7%
3	17	16.8%
4	1	1.0%
Total	101	

No. of Attributes / Entity		
Range	Frequency	%
0	43	28.5%
1 to 3	90	59.6%
4 to 6	17	11.3%
6 plus	1	0.7%

No.of Supertypes / entity		
Range	Frequency	%
0	70	46.4%
1	79	52.3%
2 plus	2	1.3%
	151	100.0%

Max path to root		
Range	Frequency	
1	39	
2	15	
3	17	
4	7	

Analysis Of Survey Data

In the survey (presented in Appendix F), the sample schema (Appendix A) to which metrics have been applied was given to students with basic but uniform EXPRESS skills. Participants were asked to locate and do a manual copying of selected schema items into a new schema. The survey involved schema items with varying levels of coupling in the form of inheritance and data abstraction as described in Chapter 4. The time taken to completely locate a type and all other types that are coupled to it was recorded. The assumption here was that, provided the search time for all types in the schema is constant and equal, the time required to locate a type and all other types that are physically dependent (coupled) on that type will be greater than the time required to locate a type with no physical dependency (coupling). The survey collected data for the following levels of coupling:

1. DAC level 0: a type is not coupled to any other type;
2. DAC level 1: a type is coupled to only one other type (in a form of physical dependency) through an attribute;
3. DAC level 2: a type is coupled to two other types (in a form of physical dependency) through an attribute;
4. DAC level 3: a type is coupled to three other types (in a form of physical dependency) through an attribute;

Similar categories were used for inheritance. An entity with inheritance level 0 (DIT=0) means the entity has no inheritance. Inheritance level 1 (DIT=1) means an entity has an inheritance with depth of one.

In the analysis of the survey data, the lower-tailed method for hypothesis testing was used to compute the difference in the mean values of the different categories of DAC and inheritance described above. The statistical method required the mean and the variance to be computed for each category or level to be compared. The computed values for the mean and variance are shown in Table 3. Using Equation 1, the test statistics z are computed and shown in Table 4.

Assuming a normal distribution, with a significant level of 0.05, the normal deviate associated with .05 significant level was found to be 1.96. This means that (using the lower-tailed method) the difference between any two mean values that is greater than or equal to -1.96 is considered significant and can be used as the basis for rejecting the Null hypothesis, H_0, (The time required to use an existing EXPRESS module does not increase significantly as coupling

between the modules increases), and accepting the alternative, H_1 (The time required to use an existing EXPRESS module increases significantly as coupling between the modules increases).

$$z = \frac{\bar{X}_a - \bar{X}_b}{\sqrt{\frac{S_a^2}{N_a} + \frac{S_b^2}{N_b}}}$$

Equation 1. Formula For Computing Normal Deviate For Comparing Means

Table 3. Mean And Variance For Different Levels Of DAC And Inheritance

DAC_ENT			Inheritance			DAC_SEL			DAC_DEF		
Level	Mean	Variance	Level	Mean	Variance	Level	Mean	Variance	Level	Mean	Variance
L0	2.85	2.99	L1	2.39	3.07	L1	3.25	17.71	L1	2.72	1.88
L1	4.05	8.37	L2	2.70	4.00	L2	4.84	8.22	L2	2.76	2.55
L2	4.30	6.95	L3	3.00	6.24	L3	4.98	16.95	L3	3.08	2.99
L3	6.20	20.04									

Table 4. Computed Test Statistic z For Different Levels Of DAC And Inheritance

DAC_ENT

M1	M2	z
L0	L1	2.8
L0	L2	3.6
L0	L3	5.5
L1	L2	0.5
L1	L3	3.2
L2	L3	2.9

Inheritance

M1	M2	Z
L1	L2	0.7
L1	L3	1.1
L2	L3	0.5

DAC_SEL

M1	M2	z
L1	L2	2.1
L1	L3	1.9
L2	L3	0.2

DAC_DEF

M1	M2	z
L1	L2	0.1
L1	L3	0.9
L2	L3	1.1

A test statistic z was computed for the mean values that were compared using the Equation 1. Tables 4 show the means that were compared and the values for the test statistic z. In Tables 4, the columns M1 and M2 denote the means to be compared. Hence, L0, L1 with a z value of 2.8 for DAC_ENT means that, 2.8 was found to be standardized difference between mean values for DAC_ENT level 0 and 1.

Observing the z values for all the comparisons, it is seen that there exists a significant difference for all the means that were compared. The value for test statistic z falls with the acceptance region of the lower-tailed test. These results support the main hypothesis H_1 and lead to the rejection of the alternative hypothesis, H_0.

CHAPTER 6
CONCLUSION

In all the categories of coupling investigated in the survey, DAC_ENT has the greatest impact on reuse of existing schema items. The time required to use a simple EXPRESS type increases at a higher rate for DAC_ENT than any other form of coupling investigated. This could be because in an EXPRESS schema, DAC_ENT can result in a recursive definition as illustrated below. A→*B*, B→*A*. Both *A* and *B* are entity types having DAC_ENT. In cases where the level of DAC is high, it becomes difficult to trace all the types that are involved in a chain of DAC. A recommendation for designing EXPRESS modules for reuse seeks a reduction of DAC_ENT in EXPRESS schemas. For instance, an attribute that uses an entity type as its data type may use a primitive type unless the attribute is composite. If that attribute uses an entity type as its data type because there is a rule in that entity, then that a rule may be migrated to a global rule. This may lead to clearer design without loss of semantics. Such a design may be easy to understand and potentially easy to reuse.

Analysis of the sample schema also reveals that very few instances of DAC_SEL existed in the schema (i.e. very few attributes used Select types as data types). Despite the minimal use of Select types as data types (DAC_SEL), the survey results show that Select types bring the second strongest form of coupling to the schema. This is seen from the increasing mean time differences between DAC_SEL values as DAC_SEL levels increased. The reason for this increasing difficulty in using items with DAC_SEL could be due to the content of the Select type definitions. Select types definitions may include other user-defined types such as Entity types, Restricted types, Enumeration types, and even other Select types. The other types that are mentioned in the Select type definition may bring other forms of coupling to the Select type making it more complicated to use.

Inheritance also brings a modest amount of coupling to the schema. Although the mean time for using entity types with different level of DIT increased as the depth of inheritance increased, the increase was not as pronounced as DAC_ENT and DAC_SEL. However, minimizing inheritance depth may improve reuse of the schema types.

In summary, this research has accomplished the following: a model has been established that predicts the reusability of EXPRESS modules. A relationship between coupling and reusability of EXPRESS modules has been shown, and a set of metrics has been developed that measure coupling in EXPRESS modules. This research has provided a foundation for further research in predicting the reusability of EXPRESS modules.

BIBLIOGRAPHY

[1] Alencar P. et al, Formal Specification of Reusable Interface Objects. *Proceedings of the 17th international conference on software engineering on Symposium on software reusability* , 1995, pp 88 - 96

[2] Bieman, James M, Kang, Byung-Kyoo. Measuring Design-Level Cohesion. *IEEE Transactions on Software Engineering*. Vol. 24, No.2, Feb. 1998

[3] Briand et al. A Unified Framework for coupling Measurement. *IEEE Transactions on Software Engineering.* vol. 25, no. 1, Jan-Feb 1998 1999.

[4] Cartwright, Michelle. An Empirical view of inheritance. Information and Sfoftware Techonolgy. Vol. 40. 1998

[5] Chidamber, Shyam, Kermerer, Chris F. *IEEE Transactions on Software Engineering.* vol. 20, no. 6, June 1994.

[6] Dai, We. Development of Reusable Components: Preliminary Experience. *Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability,* 1995, pp. 238 - 246

[7] Daniani E. et al. A Hierarchy-Aware Approach to Faceted Classification of Object-Oriented Component. *ACM Transactions on Software Engineering and Methodology*. vol. 8, no. 3, July 1999, pp. 215-262

[8] Etzkorn, Letha, Bansiya, Jagdish, Davis, Carl. Design and Code Complexity Metrics for Object-Oriented Classes. Quality Metric for Object-Oriented Design. *Journal of Object-Oriented Programming.* April 1999.

[9] Fenton, Norman E., Pfleeger, Shari L. Software Metrics: *A Rigorous & Practical Approach.* 2nd ed. PWS, 1997.

[10] Fowler, Julian. STEP for Data Management Exchange and Sharing. 1995.

[11] Frakes, W. Terry, C. *Software Reuse: Metrics and Models. ACM Computing Surveys.* vol. 28. No. 2, June 1996.

[12] Gillibrand, David, Liu, Kecheng. Quality Metric for Object-Oriented Design. *Journal of Object-Oriented Programming.* Jan 1998.

[13] Hardwick, Martin. *STEP Data Exchange Standard Moves Into Implementation Phase*. http://www.steptools.com/library/stepimpl.html

[14] Loffredo, David. *Fundamentals of STEP Implementation. http://www.steptools.com/*

[15] Li, Wei. Another Metric Suite for Object–Oriented programming. *Journal of Systems and Software*. vol. 44, Feb. 1998

[16] Mohan, Arvind; Nazemetz, John. ISO 10303 Architecture-Working of ISO/TC184/SC4/WG10 http://www.okstate.edu.ind-engr/step/WEBFILES/papers/Architecture_index.html

[17] Patel, Sukesh, Chu, William, Baxter, Rich. Measure For Composite Module Cohesion. *Proceedings of the 14th International Conference on Software Engineering*, 1992.

[18] Pressman, Roger S. Software Engineering: A Practitioner's Approach. 4th ed. McGraw-Hill, 1997

[19] Poulin et al. The business case for software reuse. IBM Systems Journal. Vol. 32, no. 4. 1993.

[20] Prieto-Diaz, Ruben. Status Report: Software Reusability. IEEE Software. May 1993.

[21] Ravat, Jayesh; Nazemetz, John. *Introduction to STEP*. *http://www.okstate.edu/ind-engr/step/WEBFILES/Papers/Introduction_index.html*

[22] Reyes, Lorna, Carver, Doris. Predicting Object Reuse Using Metrics. *Proceedings, SEKE '98: The 10th Conference on Software Engineering*, June 1998.

[23] Salil Pradihan, Nazemetz, John. STEP Goals http://www.okstate.edu/ind-engr/step/WEBFILES/Papers/Goals_index.html

[24] Shih, Timothy, Lin, Yule-Chen, Pai, Wen, Wang, Chun-Chia. Object Oriented Design Complexity based on Inheritance Relationships. *International Journal of Software Engineering.* vol. 8, no. 4, 1998.

[25] STEP Modularization Repository http://wg10step.aticorp.org/Modules/index.htm

[26] The STEP Project. http://www.nist.gov/sc4/www/stepdocs.htm

[27] Zage, M. Wayne, Zage, M. Dolores. Evaluating Design Metrics on Large-Scale Software. IEEE Software. 1993.

[28] Zhang, Jing, Warren, Thomas L. *Product Data Exchange* http://www.okstate.edu/ind-engr/step/WEBFILES/Papers/PDE_index.html

APPENDIX A

EXPRESS LISTING FOR EDITED VERSION OF AP 203(ISO-10303-203)

Appendix A provides a listing (in EXPRESS language) of the schema that was used in this research. A summary of the analysis of this schema is presented in Chapter 5. Appendix B, C, D, and E provide detailed analysis of the schema. This schema was also used in the survey (Appendix F). This schema was modified to reduce the size and complexity to fit the scope of this research.

```
(* AIM long form FOR ISO 10303-203 amendment 1
  ISO TC184/SC4/WG3 N916
   Larry McKee
  2000-05-04
*)

SCHEMA config_control_design;

 CONSTANT
 dummy_gri : geometric_representation_item := representation_item('') ||
          geometric_representation_item();
 dummy_tri : topological_representation_item := representation_item('')
          || topological_representation_item();
 END_CONSTANT;

 TYPE ahead_or_behind = ENUMERATION OF
  (ahead,
   behind);
 END_TYPE; -- ahead_or_behind

 TYPE approved_item = SELECT
  (product_definition_formation,
   product_definition,
   configuration_effectivity,
   );
 END_TYPE; -- approved_item


 TYPE Old_approved_item = SELECT  -- KOT
  (product_definition_formation,
   product_definition,
   configuration_effectivity,
   configuration_item,
   security_classification,
   change_request,
   change,
   start_request,
   start_work,
   certification,
   contract);
 END_TYPE; -- approved_item

TYPE approved_source_of_reference = SELECT
   (approved_item , certified_item);
 END_TYPE; -- axis2_placement
```

```
TYPE area_measure = REAL;
END_TYPE; -- area_measure

TYPE axis2_placement = SELECT
  (axis2_placement_2d,
   axis2_placement_3d);
END_TYPE; -- axis2_placement

TYPE b_spline_curve_form = ENUMERATION OF
  (polyline_form,
   circular_arc,
   elliptic_arc,
   parabolic_arc,
   hyperbolic_arc,
   unspecified);
END_TYPE; -- b_spline_curve_form

TYPE b_spline_surface_form = ENUMERATION OF
  (plane_surf,
   cylindrical_surf,
   conical_surf,
   spherical_surf,
   toroidal_surf,
   surf_of_revolution,
   ruled_surf,
   generalised_cone,
   quadric_surf,
   surf_of_linear_extrusion,
   unspecified);
END_TYPE; -- b_spline_surface_form

TYPE boolean_operand = SELECT
  (solid_model);
END_TYPE; -- boolean_operand

TYPE certified_item = SELECT
  (supplied_part_relationship);
END_TYPE; -- cert ified_item

TYPE change_request_item = SELECT
  (product_definition_formation);
END_TYPE; -- change_request_item

TYPE characterized_definition = SELECT
  (characterized_product_definition,
   shape_definition);
END_TYPE; -- characterized_definition

TYPE characterized_product_definition = SELECT
  (product_definition,
   product_definition_relationship);
END_TYPE; -- characterized_product_definition

TYPE classified_item = SELECT
  ( assembly_component_usage);
END_TYPE; -- classified_item

TYPE context_dependent_measure = REAL;
END_TYPE; -- context_dependent_measure

TYPE contracted_item = SELECT
  (product_definition_formation);
END_TYPE; -- contracted_item

TYPE count_measure = NUMBER;
END_TYPE; -- count_measure

TYPE curve_on_surface = SELECT
  (pcurve,
   surface_curve,
```

```
    composite_curve_on_surface);
  END_TYPE; -- curve_on_surface

  TYPE date_time_item = SELECT
    (product_definition,
     change_request,
     start_request,
     change,
     start_work,
     approval_person_organization,
     contract,
     security_classification,
     certification);
  END_TYPE; -- date_time_item

  TYPE date_time_select = SELECT
    (date,
     local_time,
     date_and_time);
  END_TYPE; -- date_time_select

  TYPE day_in_month_number = INTEGER;
  END_TYPE; -- day_in_month_number

  TYPE day_in_week_number = INTEGER;
  WHERE
    wr1: ((1 <= SELF) AND (SELF <= 7));
  END_TYPE; -- day_in_week_number

  TYPE day_in_year_number = INTEGER;
  END_TYPE; -- day_in_year_number

  TYPE descriptive_measure = STRING;
  END_TYPE; -- descriptive_measure

  TYPE dimension_count = INTEGER;
  WHERE
    wr1: (SELF > 0);
  END_TYPE; -- dimension_count

TYPE formal_approval = SELECT  (certification, approval );
END_TYPE;

  TYPE founded_item_select = SELECT
    (founded_item,
     representation_item);
  END_TYPE; -- founded_item_select

TYPE generic_definition = SELECT
  (item_definition_select);
END_TYPE;

  TYPE geometric_set_select = SELECT
    (point,
     curve,
     surface);
  END_TYPE; -- geometric_set_select

  TYPE hour_in_day = INTEGER;
  WHERE
    wr1: ((0 <= SELF) AND (SELF < 24));
  END_TYPE; -- hour_in_day

  TYPE identifier = STRING;
  END_TYPE; -- identifier

  TYPE item_definition_select = SELECT
            (product_definition_select );
  END_TYPE; -- item_definition_select
```

```
TYPE knot_type = ENUMERATION OF
  (uniform_knots,
   unspecified,
   quasi_uniform_knots,
   piecewise_bezier_knots);
END_TYPE; -- knot_type

TYPE label = STRING;
END_TYPE; -- label

TYPE length_measure = REAL;
END_TYPE; -- length_measure

TYPE list_of_reversible_topology_item = LIST [0:?] OF
         reversible_topology_item;
END_TYPE; -- list_of_reversible_topology_item

TYPE mass_measure = REAL;
END_TYPE; -- mass_measure

TYPE measure_value = SELECT
  (length_measure,
   mass_measure,
   plane_angle_measure,
   solid_angle_measure,
   area_measure,
   volume_measure,
   parameter_value,
   context_dependent_measure,
   descriptive_measure,
   positive_length_measure,
   positive_plane_angle_measure,
   count_measure);
END_TYPE; -- measure_value

TYPE minute_in_hour = INTEGER;
WHERE
  wr1: ((0 <= SELF) AND (SELF <= 59));
END_TYPE; -- minute_in_hour

TYPE month_in_year_number = INTEGER;
WHERE
  wr1: ((1 <= SELF) AND (SELF <= 12));
END_TYPE; -- month_in_year_number

TYPE parameter_value = REAL;
END_TYPE; -- parameter_value

TYPE pcurve_or_surface = SELECT
  (pcurve,
   surface);
END_TYPE; -- pcurve_or_surface

TYPE person_organization_select = SELECT
  (person,
   organization,
   person_and_organization);
END_TYPE; -- person_organization_select

TYPE plane_angle_measure = REAL;
END_TYPE; -- plane_angle_measure

TYPE positive_length_measure = length_measure;
WHERE
  wr1: (SELF > 0);
END_TYPE; -- positive_length_measure

TYPE positive_plane_angle_measure = plane_angle_measure;
WHERE
  wr1: (SELF > 0);
```

```
  END_TYPE; -- positive_plane_angle_measure

TYPE product_definition_select = SELECT
 (product_definition_formation );
END_TYPE; -- product_definition_formation

  TYPE second_in_minute = REAL;
  WHERE
    wr1: ((0 <= SELF) AND (SELF < 60));
  END_TYPE; -- second_in_minute

  TYPE set_of_reversible_topology_item = SET [0:?] OF
          reversible_topology_item;
  END_TYPE; -- set_of_reversible_topology_item

  TYPE shape_definition = SELECT
    (product_definition_shape,
     shape_aspect,
     shape_aspect_relationship);
  END_TYPE; -- shape_definition

  TYPE shell = SELECT
    (vertex_shell,
     wire_shell,
     open_shell,
     closed_shell);
  END_TYPE; -- shell

  TYPE si_prefix = ENUMERATION OF
    (exa,
     peta,
     tera,
     giga,
     mega,
     kilo,
     hecto,
     deca,
     deci,
     centi,
     milli,
     micro,
     nano,
     pico,
     femto,
     atto);
  END_TYPE; -- si_prefix

  TYPE si_unit_name = ENUMERATION OF
    (metre,
     gram,
     second,
     ampere,
     kelvin,
     mole,
     candela,
     radian,
     steradian,
     hertz,
     newton,
     pascal,
     joule,
     watt,
     coulomb,
     volt,
     farad,
     ohm,
     siemens,
     weber,
     tesla,
     henry,
```

```
  degree_celsius,
  lumen,
  lux,
  becquerel,
  gray,
  sievert);
END_TYPE; -- si_unit_name

TYPE solid_angle_measure = REAL;
END_TYPE; -- solid_angle_measure

TYPE source = ENUMERATION OF
 (made,
  bought,
  not_known);
END_TYPE; -- source

TYPE specified_item = SELECT
 (product_definition,
  shape_aspect);
END_TYPE; -- specified_item

TYPE start_request_item = SELECT
 (product_definition_formation);
END_TYPE; -- start_request_item

TYPE supported_item = SELECT
 (action_directive,
  action,
  action_method);
END_TYPE; -- supported_item

TYPE surface_model = SELECT
 (shell_based_surface_model);
END_TYPE; -- surface_model

TYPE text = STRING;
END_TYPE; -- text

TYPE transformation = SELECT
 (item_defined_transformation,
  functionally_defined_transformation);
END_TYPE; -- transformation

 TYPE unit = SELECT
 (named_unit);
END_TYPE; -- unit

TYPE vector_or_direction = SELECT
 (vector,
  direction);
END_TYPE; -- vector_or_direction

TYPE volume_measure = REAL;
END_TYPE; -- volume_measure

TYPE week_in_year_number = INTEGER;
WHERE
 wr1: ((1 <= SELF) AND (SELF <= 53));
END_TYPE; -- week_in_year_number

TYPE wireframe_model = SELECT
 (shell_based_wireframe_model,
  edge_based_wireframe_model);
END_TYPE; -- wireframe_model

TYPE work_item = SELECT
 (product_definition_formation);
END_TYPE; -- work_item
```

```
TYPE year_number = INTEGER;
END_TYPE; -- year_number

ENTITY action;
   name         : STRING;
   description  : STRING;
   chosen_method : action_method;
END_ENTITY; -- action

ENTITY action_assignment
  ABSTRACT SUPERTYPE;
   assigned_action : action;
END_ENTITY; -- action_assignment

ENTITY action_directive;
   name       : STRING;
   requests   : SET [1:?] OF versioned_action_request;
END_ENTITY; -- action_directive

ENTITY action_method;
   name       : STRING;
   description : STRING;
   consequence : STRING;
   purpose    : STRING;
END_ENTITY; -- action_method

ENTITY action_request_assignment
  ABSTRACT SUPERTYPE;
   assigned_action_request : versioned_action_request;
END_ENTITY; -- action_request_assignment

ENTITY action_request_solution;
   method  : action_method;
   request : versioned_action_request;
END_ENTITY; -- action_request_solution

ENTITY action_request_status;
   status          : label;
   assigned_request : versioned_action_request;
END_ENTITY; -- action_request_status

ENTITY action_status;
   status         : label;
   assigned_action : executed_action;
END_ENTITY; -- action_status

ENTITY address;
   internal_location      : OPTIONAL label;
   street_number          : OPTIONAL label;
   street                 : OPTIONAL label;
   postal_box             : OPTIONAL label;
   town                   : OPTIONAL label;
   region                 : OPTIONAL label;
   postal_code            : OPTIONAL label;
   country                : OPTIONAL label;
   facsimile_number       : OPTIONAL label;
   telephone_number       : OPTIONAL label;
   electronic_mail_address : OPTIONAL label;
   telex_number           : OPTIONAL label;
  WHERE
END_ENTITY; -- address

 ENTITY advanced_face
  SUBTYPE OF (face_surface);

END_ENTITY; -- advanced_face

ENTITY alternate_product_relationship;
   name       : label;
   definition : text;
```

```
    alternate  : product;
    base       : product;
    basis      : text;
  UNIQUE
    ur1 : alternate, base;
  WHERE
    wr1: (alternate :<>: base);
END_ENTITY; -- alternate_product_relationship

ENTITY application_context;
    application : text;
  INVERSE
    context_elements : SET [1:?] OF application_context_element FOR
                       frame_of_reference;
END_ENTITY; -- application_context

ENTITY application_context_element
  SUPERTYPE OF (ONEOF (product_context,product_definition_context,
      product_concept_context));
    name              : label;
    frame_of_reference : application_context;
END_ENTITY; -- application_context_element

ENTITY application_protocol_definition;
    status                              : label;
    application_interpreted_model_schema_name : label;
    application_protocol_year           : year_number;
    application                         : application_context;
END_ENTITY; -- application_protocol_definition

ENTITY approval;
    status : approval_status;
    level  : STRING;
END_ENTITY; -- approval

ENTITY approval_assignment
  ABSTRACT SUPERTYPE;
    assigned_approval : approval;
END_ENTITY; -- approval_ assignment

ENTITY approval_date_time;
    date_time          : date_time_select;
    dated_approval     : approval;
END_ENTITY; -- approval_date_time

ENTITY approval_level;
   level : STRING;
END_ENTITY; -- approval_status


ENTITY approval_person_organization;
    person_organization : person_organization_select;
    authorized_approval : approval;
    role                : approval_role;
END_ENTITY; -- approval_person_organization

ENTITY approval_relationship;
    name            : STRING;
    description     : STRING;
    relating_approval : approval;
    related_approval : approval;
    END_ENTITY; -- approval_relationship

ENTITY approval_role;
    role : label;
END_ENTITY; -- approval_role

ENTITY approval_status;
    name : label;
END_ENTITY; -- approval_status
```

```
ENTITY area_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.AREA_UNIT' IN TYPEOF(SELF\
         measure_with_unit.unit_component));
END_ENTITY; -- area_measure_with_unit

ENTITY area_unit
  SUBTYPE OF (named_unit);
END_ENTITY; -- area_unit

ENTITY assembly_component_usage
  SUPERTYPE OF (ONEOF (next_assembly_usage_occurrence,
     specified_higher_usage_occurrence, promissory_usage_occurrence))

     SUBTYPE OF (product_definition_usage);
    reference_designator : OPTIONAL identifier;
END_ENTITY; -- assembly_component_usage

ENTITY assembly_component_usage_substitute;
    name       : label;
    definition : text;
    base       : assembly_component_usage;
    substitute : assembly_component_usage;
  UNIQUE
    ur1 : base, substitute;
  WHERE
    wr1: (base.relating_product_definition :=: substitute.
         relating_product_definition);
    wr2: (base :<>: substitute);
END_ENTITY; -- assembly_component_usage_substitute

ENTITY b_spline_curve_with_knots
  SUBTYPE OF (b_spline_curve);
    knot_multiplicities : LIST  [2:?] OF INTEGER;
    knots               : LIST  [2:?] OF parameter_value;
    knot_spec           : knot_type;
  DERIVE
    upper_index_on_knots : INTEGER := SIZEOF(knots);
  WHERE
    wr1: constraints_param_b_spline(degree,upper_index_on_knots,
         upper_index_on_control_points,knot_multiplicities,knots);
    wr2: (SIZEOF(knot_multiplicities) = upper_index_on_knots);
END_ENTITY; -- b_spline_curve_with_knots

 ENTITY bounded_curve
  SUPERTYPE OF (ONEOF (polyline,b_spline_curve,trimmed_curve,
     bounded_pcurve,bounded_surface_curve,composite_curve))
  SUBTYPE OF (curve);
END_ENTITY; -- bounded_curve

ENTITY bounded_pcurve
  SUBTYPE OF (pcurve, bounded_curve);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.BOUNDED_CURVE' IN TYPEOF(SELF\pcurve.
         reference_to_curve.items[1]));
END_ENTITY; -- bounded_pcurve

ENTITY bounded_surface
  SUPERTYPE OF (ONEOF (b_spline_surface,rectangular_trimmed_surface,
     curve_bounded_surface,rectangular_composite_surface))
  SUBTYPE OF (surface);
END_ENTITY; -- bounded_surface

 ENTITY calendar_date
  SUBTYPE OF (date);
    day_component   : day_in_month_number;
    month_component : month_in_year_number;
  WHERE
```

```
   wr1: valid_calendar_date(SELF);
END_ENTITY; -- calendar_date

ENTITY cartesian_point
  SUBTYPE OF (point);
   coordinates : LIST [1:3] OF length_measure;
END_ENTITY; -- cartesian_point

ENTITY cc_design_approval
  SUBTYPE OF (approval_assignment);
   items : SET [1:?] OF approved_item;
END_ENTITY; -- cc_design_approval

ENTITY cc_design_certification
  SUBTYPE OF (certification_assignment);
   items : SET [1:?] OF certified_item;
END_ENTITY; -- cc_design_certification

ENTITY cc_design_contract
  SUBTYPE OF (contract_assignment);
   items : SET [1:?] OF contracted_item;
END_ENTITY; -- cc_design_contract

ENTITY cc_design_date_and_time_assignment
  SUBTYPE OF (date_and_time_assignment);
   items : SET [1:?] OF date_time_item;
  WHERE
   wr1: cc_design_date_time_correlation(SELF);
END_ENTITY; -- cc_design_date_and_time_assignment

ENTITY certification;
   name    : label;
   purpose : text;
   kind    : certification_type;
END_ENTITY; -- certification

ENTITY certification_assignment
  ABSTRACT SUPERTYPE;
   assigned_certification : certification;
END_ENTITY; -- certification_assignment

ENTITY certification_type;
   description : label;
END_ENTITY; -- certification_type

ENTITY change
  SUBTYPE OF (action_assignment);
   items : SET [1:?] OF work_item;
END_ENTITY; -- change

ENTITY change_request
  SUBTYPE OF (action_request_assignment);
   items : SET [1:?] OF change_request_item;
END_ENTITY; -- change_request

ENTITY chosen_action;
   action : directed_action;
END_ENTITY; -- chosen_action


ENTITY circle
  SUBTYPE OF (conic);
   radius : positive_length_measure;
END_ENTITY; -- circle

ENTITY closed_shell
  SUBTYPE OF (connected_face_set);
END_ENTITY; -- closed_shell

 ENTITY configuration_design;
```

```
    configuration : configuration_item;
    design        : product_definition_formation;
  UNIQUE
    ur1 : configuration, design;
END_ENTITY; -- configuration_design

ENTITY configuration_effectivity
  SUBTYPE OF (product_definition_effectivity);
    configuration : configuration_design;
  UNIQUE
    ur1 : configuration, usage, id;
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_USAGE' IN TYPEOF(
          SELF\product_definition_effectivity.usage));
END_ENTITY; -- configuration_effectivity

ENTITY configuration_item;
    id          : identifier;
    name        : label;
    description : OPTIONAL text;
    item_concept : product_concept;
    purpose     : OPTIONAL label;
  UNIQUE
    ur1 : id;
END_ENTITY; -- configuration_item

ENTITY conic
  SUPERTYPE OF (ONEOF (circle,ellipse,hyperbola,parabola))
  SUBTYPE OF (curve);
    position : axis2_placement;
END_ENTITY; -- conic

ENTITY conical_surface
  SUBTYPE OF (elementary_surface);
    radius     : length_measure;
    semi_angle : plane_angle_measure;
  WHERE
    wr1: (radius >= 0);
END_ENTITY; -- conical_surface

ENTITY contract;
    name    : STRING;
    purpose : STRING;
    kind    : contract_type;
END_ENTITY; -- contract

ENTITY contract_assignment
  ABSTRACT SUPERTYPE;
    assigned_contract : contract;
END_ENTITY; -- contract_assignment

ENTITY contract_type;
    description : STRING;
END_ENTITY; -- contract_type

ENTITY conversion_based_unit
  SUBTYPE OF (named_unit);
    name              : label;
    conversion_factor : measure_with_unit;
END_ENTITY; -- conversion_based_unit

ENTITY coordinated_universal_time_offset;
    hour_offset   : hour_in_day;
    minute_offset : OPTIONAL minute_in_hour;
    sense         : ahead_or_behind;
END_ENTITY; -- coordinated_universal_time_offset

ENTITY curve
  SUPERTYPE OF (ONEOF (line,conic,pcurve,surface_curve,offset_curve_3d,
    curve_replica))
```

```
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- curve

ENTITY curve_bounded_surface
  SUBTYPE OF (bounded_surface);
    basis_surface  : surface;
    boundaries     : SET [1:?] OF boundary_curve;
    implicit_outer : BOOLEAN;
END_ENTITY; -- curve_bounded_surface

  ENTITY date
  SUPERTYPE OF (ONEOF (calendar_date,ordinal_date,
     week_of_year_and_day_date));
    year_component : year_number;
END_ENTITY; -- date

ENTITY date_and_time;
    date_component : date;
    time_component : local_time;
END_ENTITY; -- date_and_time

ENTITY date_and_time_assignment
  ABSTRACT  SUPERTYPE;
    assigned_date_and_time : date_and_time;
    role                   : date_time_role;
END_ENTITY; -- date_and_time_assignment

ENTITY date_time_role;
    name : label;
END_ENTITY; -- date_time_role

ENTITY dated_effectivity
  SUBTYPE OF (effectivity);
    effectivity_start_date : date_and_time;
    effectivity_end_date   : OPTIONAL date_and_time;
END_ENTITY; -- dated_effectivity

ENTITY degenerate_pcurve
  SUBTYPE OF (point);
    basis_surface      : surface;
    reference_to_curve : definitional_representation;
  WHERE
    wr1: (SIZEOF(reference_to_curve\representation.items) = 1);
    wr2: ('CONFIG_CONTROL_DESIGN.CURVE' IN TYPEOF(reference_to_curve\
          representation.items[1]));
    wr3: (reference_to_curve\representation.items[1]\
          geometric_representation_item.dim = 2);
END_ENTITY; -- degenerate_pcurve

ENTITY directed_action
    directive : action_directive;
END_ENTITY; -- directed_action

ENTITY document;
    id          : identifier;
    name        : label;
    description : text;
    kind        : document_type;
  UNIQUE
    ur1 : id;
END_ENTITY; -- document

ENTITY document_reference
  ABSTRACT  SUPERTYPE;
    assigned_document : document;
    source            : label;
END_ENTITY; -- document_reference

ENTITY document_relationship;
    name              : label;
```

```
   description       : text;
   relating_document : document;
   related_document  : document;
END_ENTITY; -- document_relationship

ENTITY document_type;
   product_data_type : label;
END_ENTITY; -- document_type

ENTITY document_usage_constraint;
   source              : document;
   subject_element     : label;
   subject_element_value : text;
END_ENTITY; -- document_usage_constraint

ENTITY document_with_class
  SUBTYPE OF (document);
   class : identifier;
END_ENTITY; -- document_with_class

ENTITY edge
  SUPERTYPE OF (ONEOF (edge_curve,oriented_edge))
  SUBTYPE OF (topological_representation_item);
   edge_start : vertex;
   edge_end   : vertex;
END_ENTITY; -- edge

ENTITY effectivity
  SUPERTYPE OF (ONEOF (serial_numbered_effectivity,dated_effectivity,
     lot_effectivity));
   id : identifier;
END_ENTITY; -- effectivity

ENTITY elementary_surface
  SUPERTYPE OF (ONEOF (plane,cylindrical_surface,conical_surface,
     spherical_surface,toroidal_surface))
  SUBTYPE OF (surface);
   position : axis2_placement_3d;
END_ENTITY; -- elementary_surface

ENTITY ellipse;
   semi_axis_1 : positive_plane_angle_measure;
   semi_axis_2 : positive_plane_angle_measure;
END_ENTITY; -- ellipse

ENTITY evaluated_degenerate_pcurve
  SUBTYPE OF (degenerate_pcurve);
   equivalent_point : cartesian_point;
END_ENTITY; -- evaluated_degenerate_pcurve

ENTITY executed_action
  SUBTYPE OF (action);
END_ENTITY; -- executed_action

ENTITY face
  SUPERTYPE OF (ONEOF (face_surface,oriented_face))
  SUBTYPE OF (topological_representation_item);
   bounds : SET [1:?] OF face_bound;
  WHERE
   wr1: (NOT mixed_loop_type_set(list_to_set(list_face_loops( SELF))));
   wr2: (SIZEOF(QUERY ( temp <* bounds | (
         'CONFIG_CONTROL_DESIGN.FACE_OUTER_BOUND' IN TYPEOF(temp)) ))
         <= 1);
END_ENTITY; -- face

ENTITY face_bound
  SUBTYPE OF (topological_representation_item);
   bound       : loop;
   orientation : BOOLEAN;
END_ENTITY; -- face_bound
```

```
ENTITY face_outer_bound
  SUBTYPE OF (face_bound);
END_ENTITY; -- face_outer_bound

ENTITY face_surface
  SUBTYPE OF (face, geometric_representation_item);
    face_geometry : surface;
    same_sense    : BOOLEAN;

 END_ENTITY; -- faceted_brep_shape_representation

ENTITY founded_item;
END_ENTITY; -- founded_item

ENTITY geometric_representation_context
   SUBTYPE OF (representation_context);
     coordinate_space_dimension : dimension_count;
END_ENTITY; -- geometric_representation_context
ENTITY geometric_representation_item
  SUPERTYPE OF (ONEOF (point,direction,vector,placement,
     cartesian_transformation_operator,curve,surface,edge_curve,
     face_surface,poly_loop,vertex_point,solid_model,
     shell_based_surface_model,shell_based_wireframe_model,
     edge_based_wireframe_model,geometric_set))
  SUBTYPE OF (representation_item);
  DERIVE
    dim : dimension_count := dimension_of(SELF);
END_ENTITY; -- geometric_representation_item


ENTITY geometric_set
  SUPERTYPE OF (geometric_curve_set)
  SUBTYPE OF (geometric_representation_item);
   elements : SET [1:?] OF geometric_set_select;
END_ENTITY; -- geometric_set

ENTITY hyperbola
  SUBTYPE OF (conic);
    semi_axis      : positive_length_measure;
    semi_imag_axis : positive_length_measure;
END_ENTITY; -- hyperbola

ENTITY length_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.LENGTH_UNIT' IN TYPEOF(SELF\
          measure_with_unit.unit_component));
END_ENTITY; -- length_measure_with_unit

ENTITY line
  SUBTYPE OF (curve);
    pnt : cartesian_point;
    dir : vector;
  WHERE
    wr1: (dir.dim = pnt.dim);
END_ENTITY; -- line


ENTITY loop
  SUPERTYPE OF (ONEOF (vertex_loop,edge_loop,poly_lo op))
  SUBTYPE OF (topological_representation_item);
END_ENTITY; -- loop

ENTITY lot_effectivity
  SUBTYPE OF (effectivity);
    effectivity_lot_id   : identifier;
    effectivity_lot_size : measure_with_unit;
END_ENTITY; -- lot_effectivity
```

```
  ENTITY mass_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.MASS_UNIT' IN TYPEOF(SELF\
         measure_with_unit.unit_component));
END_ENTITY; -- mass_measure_with_unit

ENTITY measure_with_unit;
    value_component : measure_value;
    unit_component  : unit;
  WHERE
    wr1: valid_units( SELF);
END_ENTITY; -- measure_with_unit


ENTITY next_assembly_usage_occurrence
  SUBTYPE OF (assembly_component_usage);
END_ENTITY; -- next_assembly_usage_occurrence

ENTITY organization;
    id          : OPTIONAL identifier;
    name        : label;
    description : text;
END_ENTITY; -- organization

 ENTITY parabola
  SUBTYPE OF (conic);
    focal_dist : length_measure;
  WHERE
    wr1: (focal_dist <> 0);
END_ENTITY; -- parabola

ENTITY parametric_representation_context
  SUBTYPE OF (representation_context);
END_ENTITY; -- parametric_representation_context

ENTITY path
  SUPERTYPE OF (ONEOF (edge_loop,oriented_ path))
  SUBTYPE OF (topological_representation_item);
    edge_list : LIST  [1:?] OF UNIQUE oriented_edge;
  WHERE
    wr1: path_head_to_tail(SELF);
END_ENTITY; -- path

ENTITY pcurve
  SUBTYPE OF (curve);
    basis_surface      : surface;
    reference_to_curve : definitional_representation;
  WHERE
    wr1: (SIZEOF(reference_to_curve\representation.items) = 1);
    wr2: ('CONFIG_CONTROL_DESIGN.CURVE' IN TYPEOF(reference_to_curve\
         representation.items[1]));
    wr3: (reference_to_curve\representation.items[1]\
         geometric_representation_item.dim = 2);
END_ENTITY; -- pcurve

ENTITY person;
    id          : identifier;
    last_name     : OPTIONAL label;
    first_name    : OPTIONAL  label;
    middle_names  : OPTIONAL LIST [1:?] OF label;
    prefix_titles : OPTIONAL LIST  [1:?] OF label;
    suffix_titles : OPTIONAL LIST [1:?] OF label;
  UNIQUE
    ur1 : id;
  WHERE
    wr1: (EXISTS(last_name) OR EXISTS(first_name));
END_ENTITY; -- person

ENTITY person_and_organization;
```

```
  the_person      : person;
  the_organization : organization;
END_ENTITY; -- person_and_organization

ENTITY person_and_organization_assignment
 ABSTRACT SUPERTYPE;
  assigned_person_and_organization : person_and_organization;
  role                             : person_and_organization_role;
END_ENTITY; -- person_and_organization_assignment

ENTITY person_and_organization_role;
  name : label;
END_ENTITY; -- person_and_organization_role

ENTITY personal_address
 SUBTYPE OF (address);
  people      : SET [1:?] OF person;
  description : text;
END_ENTITY; -- personal_address

ENTITY placement
 SUPERTYPE OF (ONEOF (axis1_placement,axis2_placement_2d,
    axis2_placement_3d))
 SUBTYPE OF (geometric_representation_item);
  location : cartesian_point;
END_ENTITY; -- placement

ENTITY plane
 SUBTYPE OF (elementary_surface);
END_ENTITY; -- plane

ENTITY plane_angle_measure_with_unit
 SUBTYPE OF (measure_with_unit);
 WHERE
  wr1: ('CONFIG_CONTROL_DESIGN.PLANE_ANGLE_UNIT' IN TYPEOF(SELF\
       measure_with_unit.unit_component));
END_ENTITY; -- plane_angle_measure_with_unit

ENTITY point
 SUPERTYPE OF (ONEOF (cartesian_point,point_on_curve,point_on_surface,
    point_replica,degenerate_pcurve))
 SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- point

ENTITY point_on_curve
 SUBTYPE OF (point);
  basis_curve     : curve;
  point_parameter : parameter_value;
END_ENTITY; -- point_on_curve

ENTITY product;
  id                : STRING;
  name              : STRING;
  description       : STRING;
  frame_of_reference : SET [1:?] OF product_context;
 UNIQUE
  ur1 : id;
END_ENTITY; -- product

ENTITY product_category;
  name       : STRING;
  description : OPTIONAL STRING;
END_ENTITY; -- product_category

ENTITY product_category_relationship;
  name                : label;
  description         : text;
  category            : product_category;
  sub_category        : product_category;
 WHERE
```

```
   wr1: acyclic_product_category_relationship(SELF,[SELF.sub_category]);
END_ENTITY; -- product_category_relationship

ENTITY product_concept;
   id                  : STRING;
   name                : STRING;
   description         : STRING;
   market_context      : product_concept_context;
 UNIQUE
   ur1 : id;
END_ENTITY; -- product_concept

ENTITY product_concept_context;
   market_segment_type : STRING;
END_ENTITY; -- product_concept_context

ENTITY product_context
  SUBTYPE OF (application_context_element);
   discipline_type : label;
END_ENTITY; -- product_context

ENTITY product_definition;
   id              : identifier;
   description     : text;
   formation       : product_definition_formation;
   frame_of_reference : product_definition_context;
END_ENTITY; -- product_definition

ENTITY product_definition_context
  SUBTYPE OF (application_context_element);
   life_cycle_stage : label;
END_ENTITY; -- product_definition_context

ENTITY product_definition_effectivity
  SUBTYPE OF (effectivity);
   usage : product_definition_relationship;
 UNIQUE
   ur1 : usage, id;
END_ENTITY; -- product_definition_effectivity

ENTITY product_definition_formation;
   id          : STRING;
   description : SRING;
END_ENTITY; -- product_definition_formation


ENTITY product_definition_relationship;
   id                      : identifier;
   name                    : label;
   description             : text;
   relating_product_definition : product_definition;
   related_product_definition  : product_definition;
END_ENTITY; -- product_definition_relationship

ENTITY product_definition_shape
  SUBTYPE OF (property_definition);
 UNIQUE
   ur1 : definition;
 WHERE
   wr1: (NOT ('CONFIG_CONTROL_DESIGN.SHAPE_DEFINITION' IN TYPEOF(SELF\
         property_definition.definition)));
END_ENTITY; -- product_definition_shape

ENTITY product_definition_usage
  SUPERTYPE OF (assembly_component_usage)
  SUBTYPE OF (product_definition_relationship);
 UNIQUE
   ur1 : id, relating_product_definition, related_product_definition;
 WHERE
   wr1: acyclic_product_definition_relationship(SELF,[SELF\
```

```
            product_definition_relationship.related_product_definition],
            'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_USAGE');
  END_ENTITY; -- product_definition_usage

ENTITY product_related_product_category;
            the_product: product;
END_ENTITY; -- product_related_product_category

  ENTITY property_definition;
     name       : label;
     description : text;
     definition  : characterized_definition;
  END_ENTITY; -- property_definition

  ENTITY property_definition_representation;
     definition          : property_definition;
     used_representation : representation;
  END_ENTITY; -- property_definition_representation

  ENTITY representation;
     name            : label;
     items           : SET [1:?] OF representation_item;
     context_of_items : representation_context;
  END_ENTITY; -- representation

  ENTITY representation_context;
     context_identifier : identifier;
     context_type       : text;
   INVERSE
     representations_in_context : SET [1:?] OF representation FOR
                                  context_of_items;
  END_ENTITY; -- representation_context

  ENTITY representation_item;
     name : label;
   WHERE
     wr1: (SIZEOF(using_representations(SELF)) > 0);
  END_ENTITY; -- representation_item

  ENTITY security_classification_level;
     name : label;
  END_ENTITY; -- security_classification_level

  ENTITY serial_numbered_effectivity
    SUBTYPE OF (effectivity);
     effectivity_start_id : identifier;
     effectivity_end_id   : OPTIONAL identifier;
  END_ENTITY; -- serial_numbered_effectivity

  ENTITY shape_aspect;
     name              : label;
     description       : text;
     of_shape          : product_definition_shape;
     product_definitional : LOGICAL;
  END_ENTITY; -- shape_aspect


  ENTITY shape_definition_representation
    SUBTYPE OF (property_definition_representation);
   END_ENTITY; -- shape_definition_representation

  ENTITY shape_representation
    SUBTYPE OF (representation);
  END_ENTITY; -- shape_representation

  ENTITY shape_representation_relationship
    SUBTYPE OF (representation_relationship);
    WHERE
      wr1: ('CONFIG_CONTROL_DESIGN.SHAPE_REPRESENTATION' IN (TYPEOF(SELF\
            representation_relationship.rep_1) + TYPEOF(SELF\
```

```
          representation_relationship.rep_2)));
END_ENTITY; -- shape_representation_relationship


ENTITY si_unit
  SUBTYPE OF (named_unit);
    prefix : OPTIONAL  si_prefix;
    name   : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
                          dimensions_for_si_unit(SELF.name);
END_ENTITY; -- si_unit

ENTITY solid_angle_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.SOLID_ANGLE_UNIT' IN TYPEOF(SELF\
          measure_with_unit.unit_component));
END_ENTITY; -- solid_angle_measure_with_unit


ENTITY solid_model
  SUPERTYPE OF (manifold_solid_brep)
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- solid_model

ENTITY specified_higher_usage_occurrence
  SUBTYPE OF (assembly_component_usage);
    upper_usage : assembly_component_usage;
   next_usage  : next_assembly_usage_occurrence;
  UNIQUE
    ur1 : upper_usage, next_usage;
END_ENTITY; -- specified_higher_usage_occurrence

ENTITY spherical_surface
  SUBTYPE OF (elementary_surface);
    radius : positive_length_measure;
END_ENTITY; -- spherical_surface

ENTITY start_request
  SUBTYPE OF (action_request_assignment);
    items : SET  [1:?] OF start_request_item;
END_ENTITY; -- start_request

ENTITY start_work
  SUBTYPE OF (action_assignment);
    items : SET  [1:?] OF work_item;
END_ENTITY; -- start_work

ENTITY supplied_part_relationsh ip;
END_ENTITY; -- supplied_part_relationship

ENTITY surface
  SUPERTYPE OF (ONEOF (elementary_surface,swept_surface,bounded_surface,
     offset_surface,surface_replica))
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- surface


ENTITY surface_replica
  SUBTYPE OF (surface);
    parent_surface : surface;
    transformation : cartesian_transformation_operator_3d;
  WHERE
    wr1: acyclic_surface_replica(SELF,parent_surface);
END_ENTITY; -- surface_replica


ENTITY topological_representation_item
  SUPERTYPE OF (ONEOF (vertex,edge,face_bound,face,vertex_shell,
```

```
      wire_shell,connected_edge_set,connected_face_set,loop ANDOR path))
  SUBTYPE OF (representation_item);
END_ENTITY; -- topological_representation_item

ENTITY toroidal_surface
  SUBTYPE OF (elementary_surface);
    major_radius : positive_length_measure;
    minor_radius : positive_length_measure;
END_ENTITY; -- toroidal_surface


ENTITY uniform_curve
  SUBTYPE OF (b_spline_curve);
END_ENTITY; -- uniform_curve

ENTITY uniform_surface
  SUBTYPE OF (b_spline_surface);
END_ENTITY; -- uniform_surface

ENTITY valid_reference_source;
 source: approved_source_of_reference;
END_ENTITY; -- uniform_surface


ENTITY vector
  SUBT YPE OF (geometric_representation_item);
    orientation : direction;
    magnitude   : length_measure;
  WHERE
    wr1: (magnitude >= 0);
END_ENTITY; -- vector

ENTITY versioned_action_request;
    id          : STRNG;
    version     : STRING;
END_ENTITY; -- versioned_action_request

ENTITY vertex
  SUBTYPE OF (topological_representation_item);
END_ENTITY; -- vertex

ENTITY vertex_loop
  SUBTYPE OF (loop);
    loop_vertex : vertex;
END_ENTITY; -- vertex_loop

ENTITY vertex_point
  SUBTYPE OF (vertex, geometric_representation_item);
    vertex_geometry : point;
END_ENTITY; -- vertex_point

END_SCHEMA; – config_control_design
```

APPENDIX B

ANALYSIS OF SCHEMA AP 203 (ENTITY TYPES)

This Appendix provides the result of the analysis of schema in Appendix A. In this Appendix, the analysis of Entity types and Inheritance is presented. In the table below, the first and second columns tell if the entity is a supertype or subtype respectively. The third column, *Root* tells whether or not the entity is a root supertype. *Num Sub* and *Num Super* give the number of subtypes and supertypes respectively for that entity. *Max DIT* is the longest path from the entity to its subtypes. *Max Super_Path* is longest path from the entity to its supertypes. *Max DAC_Path* is the longest path (for attributes in this in the entity) from an attribute to its underlying type.

	super	sub		Num	Num	Max	Max	Max
Entity	type?	type?	Root?	Sub	Super	DIT	Super_Path	DAC_Path
action	Y	N	Y	1	0	1	0	1
action_assigment	Y	N	Y	1	0	1	0	0
action_directive	N	N	N/A	0	0	0	0	1
action_method	N	N	N/A	0	0	0	0	0
action_request_assignment	Y	N	Y	2	0	0	0	1
action_request_solution	N	N	N/A	0	0	0	0	1
action_request_status	N	N	N/A	0	0	0	0	1
action_status	N	N	N/A	0	0	0	0	1
address	Y	N	Y	1	0	1	0	1
advanced_face	N	Y	N/A	0	1	0	4	0
alternate_product_relationship	N	N	N/A	0	0	0	0	2
application_context	N	N	N/A	0	0	0	0	3
application_context_element	Y	N	Y	3	0	1	0	2
application_protocol_definition	N	N	N/A	0	0	0	0	3
approval	N	N	N/A	0	0	0	0	2
approval_assignment	Y	N	Y	1	0	1	0	3
approval_date_time	N	N	N/A	0	0	0	0	3
approval_level	N	N	N/A	0	0	0	0	0
approval_person_organization	N	N	N/A	0	0	0	0	3
approval_relationship	N	N	N/A	0	0	0	0	3
approval_role	N	N	N/A	0	0	0	0	1
approval_status	N	N	N/A	0	0	0	0	1
area_measure_with_unit	N	Y	N/A	0	1	0	1	0
area_unit	N	Y	N/A	0	1	0	1	0
assembly_component_usage	Y	Y	N	3	1	1	2	1
assembly_component_usage_substitute	N	N	N/A	0	0	0	0	2
b_spline_curve_with_knots	N	Y	N/A	1	0	0	1	1

Entity	super type?	sub type?	root?	Num Sub	Num Super	Max DIT	MAX Super_Path	Max DAC Path
bounded_curve	Y	Y	N	5	1	1	3	0
bounded_pcurve	N	Y	N/A	0	2	0	4	0
bounded_surface	Y	Y	N	4	1	1	3	0
calendar_date	N	Y	N/A	0	1	0	1	1
cartesian_point	N	Y	N/A	0	1	0	3	1
cc_design_approval	N	Y	N/A	0	1	0	1	1
cc_design_certification	N	Y	N/A	0	1	0	1	1
cc_design_contract	N	Y	N/A	0	1	0	1	1
cc_design_date_and_time_assignment	N	Y	N/A	0	1	0	1	1
certification	N	N	N/A	0	0	0	0	2
certification_assignment	Y	N	Y	1	0	2	0	3
certification_type	N	N	N/A	0	0	0	0	1
change	N	Y	N/A	0	1	0	1	1
change_request	N	Y	N/A	0	1	0	1	1
chosen_action	N	N	N/A	0	0	0	0	3
circle	N	Y	N/A	0	1	0	3	2
closed_shell	N	Y	N/A	0	1	0	1	0
configuration_design	N	N	N/A	0	0	0	0	3
configuration_effectivity	N	Y	N/A	0	1	0	2	4
configuration_item	N	N	N/A	0	0	0	0	2
conic	Y	Y	N	4	1	1	2	2
conical_surface	N	Y	N/A	0	1	0	4	0
contract	N	N	N/A	0	0	0	0	1
contract_assignment	Y	N	Y	1	0	3	0	1
contract_type	N	N	N/A	0	0	0	0	1
conversion_based_unit	N	Y	N/A	0	1	0	1	1
coordinated_universal_time_offset	N	N	N/A	0	0	0	0	1
curve	Y	Y	N	5	1	1	2	0
curve_bounded_surface	N	Y	N/A	0	1	0	0	3
date	Y	N	Y	3	0	1	0	1
date_and_time	N	N	N	0	0	0	0	2
date_and_time_assignment	Y	N	Y	1	0	0	0	3
date_time_role	N	N	N/A	0	0	0	0	1
dated_effectivity	N	Y	N/A	0	1	0	1	3
degenerate_pcurve	N	Y	N/A	0	1	0	3	2
directed_action	N	N	N/A	0	0	0	0	1
document	N	N	N/A	0	0	0	0	1
document_reference	N	N	N/A	0	0	0	0	2
document_relationship	N	N	N/A	0	0	0	0	2
document_type	N	N	N/A	0	0	0	0	1
document_usage_constraint	N	N	N/A	0	0	0	0	3
document_with_class	N	Y	N/A	0	1	0	1	1
edge	Y	Y	N	2	1	1	2	2
effectivity	Y	N	Y	3	0	1	0	1
elementary_surface	Y	Y	N	5	1	1	3	1
ellipse	N	N	N/A	0	0	0	0	1

	super	sub		Num	Num	Max	MAX	Max
Entity	type?	type?	root?	Sub	Super	DIT	Super_Path	DAC Path
evaluated_degenerate_pcurve	N	Y	N/A	0	1	0	4	3
executed_action	N	Y	N/A	0	1	0	1	0
face	Y	Y	N	2	1	1	2	2
face_bound	N	Y	N/A	0	1	0	2	1
face_outer_bound	N	Y	N/A	0	1	0	3	0
face_surface	N	Y	N/A	0	1	0	3	3
founded_item	N	N	N/A	0	0	0	0	0
geometric_representation_context	N	Y	N/A	0	1	0	1	1
geometric_representation_item	Y	Y	N	15	1	3	1	1
geometric_set	Y	Y	N	1	1	1	1	1
hyperbola	N	Y	N/A	0	1	0	3	2
length_measure_with_unit	N	Y	N/A	0	1	0	1	3
line	N	Y	N/A	0	1	0	3	3
loop	Y	Y	N	3	1	1	2	0
lot_effectivity	N	Y	N/A	0	1	0	1	2
mass_measure_with_unit	N	Y	N/A	0	1	0	1	0
measure_with_unit	N	N	N/A	0	0	0	0	1
next_assembly_usage_occurrence	N	Y	N/A	0	1	0	3	0
organization	N	N	N/A	0	0	0	0	1
parabola	N	Y	N/A	0	1	0	3	1
parametric_representation_context	N	Y	N/A	0	1	0	1	0
path	Y	Y	N	2	1	1	2	1
pcurve	N	Y	N/A	0	1	0	3	1
person	N	N	N	0	0	0	0	1
person_and_organization	N	N	N	0	0	0	0	2
person_and_organization_assignment	N	N	N/A	0	0	0	0	3
person_and_organization_role	N	N	N/A	0	0	0	0	1
personal_address	N	Y	N/A	0	1	0	1	2
placement	Y	Y	N	2	1	1	1	3
plane	N	Y	N/A	0	1	0	4	0
plane_angle_measure_with_unit	N	Y	N/A	0	1	0	1	0
point	Y	Y	N	1	5	1	2	0
point_on_curve	N	Y	N/A	0	1	0	3	3
product	N	N	N/A	0	0	0	0	2
product_category	N	N	N/A	0	0	0	0	0
product_category_relationship	N	N	N/A	0	0	0	0	1
product_concept	N	N	N/A	0	0	0	0	0
product_concept_context	N	N	N/A	0	0	0	0	1
product_context	N	Y	N/A	0	1	0	1	0
product_definition	N	N	N/A	0	0	0	0	1
product_definition_context	N	Y	N/A	0	1	0	1	1
product_definition_effectivity	N	Y	N/A	0	1	0	1	1
product_definition_formation	N	N	N/A	0	0	0	0	0
product_definition_relationship	N	N	N/A	0	0	0	0	1
product_definition_shape	N	Y	N/A	0	1	0	1	0

Entity	super type?	sub type?	root?	Num Sub	Num Super	Max DIT	MAX Super_Path	Max DAC Path
product_definition_usage	Y	Y	N/A	1	1	2	1	0
product_related_product_category	N	N	N/A	0	0	0	0	2
product_definition	N	N	N/A	0	0	0	0	1
product_definition_representation	N	N	N/A	0	0	0	0	2
representation	N	N	N/A	0	0	0	0	2
representation_context	N	N	N/A	0	0	0	0	2
representation_item	N	N	N/A	0	0	0	0	1
security_classification_level	N	N	N/A	0	0	0	0	1
serial_numbered_effectivity	N	Y	N/A	0	1	0	1	1
shape_aspect	N	N	N/A	0	0	0	0	1
shape_definition_representation	N	Y	N/A	0	1	0	1	0
shape_representation	N	Y	N/A	0	1	0	1	0
shape_representation_relationship	N	Y	N/A	0	1	0	2	0
si_unit	N	Y	N/A	0	1	0	1	0
solid_angle_measure_with_unit	N	Y	N/A	0	1	0	1	0
solid_model	Y	Y	N	1	1	1	1	0
specified_higher_usage_occurrence	N	Y	N/A	0	1	0	3	2
spherical_surface	N	Y	N/A	0	1	0	4	2
start_request	N	Y	N/A	0	1	0	1	0
start_work	N	Y	N/A	0	1	0	1	0
supplied_part_relationship	N	N	N/A	0	0	0	0	0
surface	Y	Y	N	5	1	2	2	0
surface_replica	N	Y	N/A	0	1	0	3	2
topological_representation_item	Y	Y	N	10	1	3	1	0
toroidal_surface	N	Y	N/A	0	1	0	4	2
uniform_curve	N	Y	N/A	0	1	0	1	0
uniform_surface	N	Y	N/A	0	1	0	4	0
valid_reference_source	N	N	N/A	0	0	0	0	1
vector	N	Y	N/A	0	1	0	2	0
versioned_action_request	N	N	N/A	0	0	0	0	1
vertex	N	Y	N/A	0	1	0	2	0
vertext_loop	N	Y	N/A	0	1	0	3	2
vertext_point	N	Y	N/A	0	1	0	2	0

APPENDIX C

ANALYSIS OF SCHEMA AP 203 (ATTRIBUTE TYPES)

This Appendix provides the result of the analysis of the Attributes of the Entities of the schema in Appendix A. The table shows an entity and its attributes in the first column, and the underlying type in the second column. The third column assigns a code that describes the underlying type of the attribute. DAC_PATH again shows, for each attribute, the longest path from that attribute to its underlying type.

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
action			
name	STRING	B	0
description	STRING	B	0
chosen_method	action_method	T	1
action_assigment			
assigned_action	action	E	0
action_directive			
name	STRING	B	0
request	versioned_action_request	T	1
action_method			
name	STRING	B	0
descritpion	STRING	B	0
consequence	STRING	B	0
purpose	STRING	B	0
action_request_assignment			
assigned_action_request	versioned_action_request	T	1
action_request_solution			
method	action_method	T	1
request	versioned_action_request	T	1
action_request_status			
status	label	R	1
assigned_request	versioned_action_request	T	1
action_status			
status	label	R	1
assigned_action	executed_action	T	1

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
address			
internal_location	label	R	1
street_number	label	R	1
street	label	R	1
postal_box	label	R	1
town	label		1
region	label	R	1
postal_code	label	R	1
country	label	R	1
facsmile_number	label	R	1
telephone_number	label	R	1
electronic_mail_address	label	R	1
telex_number	label	R	1
advanced_face			
alternate_product_relationship			
name	label	R	1
definition	text	R	1
alternate	product	T	2
base	product	T	2
basis	text	R	1
application_context			
application	text	T	3
application_context_element			
name	label	R	1
frame_of_reference	application_context	T	2
application_protocol_definition			
status	label	R	1
application_interpreted_model_schema_name	label	R	1
application_protocol_year	year_number	R	1
application	aplication_context	T	3
approval			
status	approval_status	T	2
level	STRING	B	0
approval_assignment			
assigned_approval	approval_status	T	3
approval_date_time			
date_time	date_time_select	S	1
dated_aproval	approval	T	3
approval_level			
level	STRING	B	0

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
approval_person_organization			
person_organization	person_organization_select	T	1
authorized_approval	approval	T	3
role	approval_role	T	2
approval_relationship			
name	STRING	B	0
description	STRING	B	0
relating_approval	approval	T	3
related_approval	approval	T	3
approval_role			
role	label	R	1
approval_status			
name	label	R	1
area_measure_with_unit			N/A
area_unit			N/A
assembly_component_usage			
reference_designator	identifier	R	1
assembly_component_usage_substitute			
name	label	R	1
definition	text	R	1
base	assembly_component_usage	T	2
substitute	assembly_component_usage	T	2
b_spline_curve_with_knots			
knot_multiplicities	INTEGER	B	0
knots	pamater_value	R	1
knot_spec	knot_type	T	1
bounded_curve			
bounded_pcurve			N/A
bounded_surface			N/A
calendar_date			N/A
day_component	day_in_month_number	R	1
month_component	month_in_year_number	R	1
cartesian_point			
coordinates	length_measure	R	1
cc_design_approval			
items	approved_item	S	1
cc_design_certification			
items	certified_item	S	1

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
cc_design_contract			
items	contract_item	S	1
cc_design_date_and_time_assignment			
items	date_time_item	T	1
Certification			
name	label	R	1
purpose	text	R	1
kind	certification_type	T	2
certification_assignment			
assigned_certification	certification	T	3
certification_type			
description	label	R	1
change			
items	work_item	T	1
change_request			
items	change_request_item	T	1
chosen_action			
action	directed_action	T	3
circle			
readius	positive_length_measure	R	2
closed_shell			N/A
configuration_design			
configuration	configuration_item	T	3
design	product_defintition_formation	T	1
configuration_effectivity			
configuration	configuration_design	T	4
configuration_item			
id	identifier	R	1
name	label	R	1
description	text	R	1
item_concept	product_concept	T	2
purpose	label	R	1
conic			
position	axis2_placement	T	2
conical_surface			
radius	length_measure	B	0
semi_angle	plane_angle_measure	B	0

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
contract			
name	STRING	B	0
purpose	STRING	B	0
kind	contract_type	T	1
contract_assignment			
assigned_contract	contract	T	1
contract_type			
description	STRING	B	0
conversion_based_unit			
name	label	R	1
conversion_factor	measure_with_unit	T	1
coordinated_universal_time_offset			
hour_offset	hour_in_day	R	1
minute_offset	minute_in_day	R	1
sense	ahead_or_behind	E	1
curve			N/A
curve_bounded_surface			
basis_surface	surface	T	3
boundaries	boundary_curve	T	1
implicit_outer	BOOLEAN	B	0
date			
year_component	year_number	R	1
date_and_time			
date_component	date	T	2
time_component	local_time	T	1
date_and_time_assignment			
assigned_date_abd_time	date_and_time	T	3
role	date_time_role	T	3
date_time_role			1
name	label	R	1
dated_effectivity			
effective_start_date	date_and_time	T	3
effective_end_date	date_and_time	T	3
degenerate_pcurve			
basis	surface	T	2
reference_to_curve	defintional_representation	T	1
directed_action			

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
directive	action_directive	T	1
document			
id	identifier	R	1
name	label	R	1
description	text	R	1
kind	document_type	T	1
document_reference			
assigned_document	document	T	2
source	label	R	1
document_relationship			
name	label	R	1
description	text	R	1
relating_document	document	T	2
related_document	document	T	2
document_type			
product_data_type	label		1
document_usage_constraint			
source	document	T	2
subject_element	label	R	1
subject_element_value	text	R	1
document_with_class			
class	identifier	R	1
edge			
edge_start	vertext	T	1
edge_end	vertext	T	1
effectivity			
id	identifier	R	1
elementary_surface			
position	axis2_placement_3d	T	1
ellipse			
semi_axis_1	positive_plane_angle_measure	T	1
semi_axis_2	positive_plane_angle_measure	T	1
evaluated_degenerate_pcurve			
equivalent_point	cartesian_point	T	1
executed_action			N/A

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
Face			
bounds	face_bound	T	1
Face_bound			
bound	loop	T	1
orientation	BOOLEAN	B	0
Face_outer_bound			N/A
Face_surface			
Face_goemetry	surface	T	1
founded_item			N/A
geometric_representation_context			
coordinate_space_dimension	dimension_count	T	1
geometric_representation_item			
dim	dimension_count	T	1
geometric_set			
elements	geometric_set_select	T	1
hyperbola			
semi_axis	positive_length_measure	R	2
semi_imag_axis	positive_length_measure	R	2
length_measure_with_unit			N/A
Line			
pnt	cartesian_point	T	1
dir	vector	T	1
Loop			N/A
lot_effectivity			
effectivity_lot_id	identifier	R	1
effectivity_lot_size	measure_with_unit	T	2
mass_measure_with_unit			N/A
measure_with_unit			
value_component	measure_value	R	1
unit_component	unit	T	1
Next_assembly_usage_occurrence			N/A
organization			
id	identifier	R	1
name	label	R	1
description	text	R	1
parabola			
focal_dist	length_measure	R	1

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
parametric_representation_context			
path			N/A
edge_list	orientation_edge	T	1
pcurve			
basis_surface	surface	T	1
reference_to_curve	definitional_representation	T	1
person			
id	identifier	R	1
last_name	label	R	1
first_name	label	R	1
middle_name	label	R	1
prefix_titles	label	R	1
suffix_titles	label	R	1
person_and_organization			
the_person	person	T	2
the_organization	organization	T	2
person_and_organization_assignment			
assigned_person_and_organization	person_and_organization	T	3
role	person_organization_role	T	2
person_and_organization_role			
name	label	R	1
personal_address			
people	person	T	2
description	text	R	1
placement			
location	cartesian_point	T	1
plane			N/A
plane_angle_measure_with_unit			N/A
point			N/A
point_on_curve			
basis_curve	curve	T	1
point_parameter	parameter_value	R	1
product			
id	STRING	B	0
name	STRING	B	0
description	STRING	B	0
frame_of_reference	product_context	T	2

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
product_category			
name	STRING	B	0
description	STRING	B	0
product_category_relationship			
name	label	R	1
description	text	R	1
category	product_category	T	1
sub_category	product_category	T	1
product_concept			
id	identifier	B	0
name	label	B	0
description	text	B	0
market_context	product_concept_context	T	1
product_concept_context			
market_segment_type	STRING	B	0
product_context			
descipline_type	label	R	1
product_definition			
id	identifier	R	1
description	text	R	1
formation	product_definition_formation	T	1
frame_of_reference	product_definition_context	T	1
product_definition_context			
life_cycle_stage	label	R	1
product_definition_effectivity			
usage	product_definition_usage	T	1
product_definition_formation			
id	STRING	B	0
description	STRING	B	0
product_definition_relationship			
id	identifier	R	1
name	label	R	1
description	text	R	1
relating_product_definition	product_definition	T	Q

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
related_product_definition	product_definition	T	2
product_definition_shape			N/A
product_definition_usage			N/A
product_related_product_category			
the_product	product	T	2
product_definition			
name	label	R	1
description	text	R	1
definition	characterized_definition	T	1
product_definition_representation			
definition	property_definition	T	1
used_representation	representation	T	2
representation			
name	label	R	1
items	representation_item	T	1
context_of_items	representation_context	T	2
representation_context			
context_identifier	identifier	R	1
context_type	text	R	1
representation_in_context	representation	T	2
representation_item			
name	label	R	1
security_classification_level			
name	label	R	1
serial_numbered_effectivity			
effectivity_start_date	identifier	R	1
effectivity_end_date	identifier	R	1
shape_aspect			
name	label	R	1
description	text	R	1
of_shape	product_definition_shape	T	1
product_definitional	LOGICAL	B	0
shape_definition_representation			N/A
shape_representation			N/A
shape_representation_relationship			N/A
si_unit			
prefix	si_prefix	E	0

Entity /Attributes	Underlying Type Name	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	DAC_Path
name	si_unit_name	E	0
solid_angle_measure_with_unit			
solid_model			
specified_higher_usage_occurrence			
upper_usage	assembly_component_usage	T	2
next_usage	next_assembly_usage_occurrence	T	1
spherical_surface			
raduis	positive_length_measure	R	2
start_request			
items	start_request_item	T	1
start_work			
items	work_item	T	1
supplied_part_relationship			
surface			N/A
surface_replica			N/A
parent_surface	surface	T	2
transformation	cartesian_transformation_operator_3d	T	1
topological_representation_item			
toroidal_surface			
major_radius	positive_length_measure	R	2
minor_radius	positive_length_measure	R	2
uniform_curve			N/A
uniform_surface			N/A
valid_reference_source			
source	approved_source_of_reference	S	1
vector			
orientation	direction	E	0
magnitude	length_measure	R	1
versioned_action_request			
id	STRING	B	0
version	STRING	B	0
vertex			
vertext_loop	vertex		1
vertext_point			
vertex_geometry	point	T	2

APPENDIX D

ANALYSIS OF SCHEMA AP 203 (SELECT TYPES)

This Appendix provides the result of the analysis of the Select types of schema in Appendix A. The first column shows, for each Select type, the number of Select list items, the second column shows the type composition of the Select list. For instance, if the Select type contains entities in the Select list, then the code T is assigned. The last column shows how many times that type has been redefined.

Number of Select list items	Composition (E = Enum, R = Restricted types, S = Select Types, T = Entities, M = Mixed)	Maximum Level of definition
3	T	1
11	T	1
2	S	2
2	E	1
1	T	2
1	T	1
2	T	1
1	T	2
1	T	1
3	T	2
9	T	1
3	T	1
2	T	1
2	T	1
1	S	2
3	T	3
1	S	2
12	R	1
2	T	2
3	T	1
1	T	1
3	T	1
2	T	1
1	T	1
3	T	1
1	T	1
2	T	1
1	T	1
2	T	1
1	T	1
2	T	1

APPENDIX E

ANALYSIS OF SCHEMA AP 203 (RESTRICTED TYPES)

This Appendix provides the result of the analysis of the Restricted types of the schema in Appendix A. The columns show the type name, the underlying type and the longest path to the underlying type (Max DAC)

	Underlying type (E = Enum, R = Restricted types, S = Select Types, B = Base type, T = Entity)	Max DAC
Type Name		0
area_measure	B	0
context_dependent_measure	B	0
context_dependent_measure	B	0
count_measure	B	0
day_in_month_number	B	0
day_in_week_number	B	0
day_in_year_number	B	0
descriptive_measure	B	0
dimension_count	B	0
hour_in_day	B	0
identifier	B	0
label	B	0
length_measure	B	0
list_of_reversible_topology_item	T	1
mass_measure	B	0
minute_in_hour	B	0
month_in_year_number	B	0
parameter_value	B	0
plane_angle_measure	B	0
positive_length_measure	R	1
positive_plane_angle_measure	R	1
second_in_minute	B	0
set_of_reversible_topology_item	T	0
solid_angle_measure	B	1
text	B	0
volume_measure	B	0
week_in_year_number	B	0
year_number	B	0

APPENDIX F

SURVEY

This Appendix presents the survey instrument that was used to obtain information about the reuse of EXPRESS modules. The survey was given to undergraduate students with one semester training in EXPRESS. The survey required participants to reuse existing EXPRESS modules in the schema provided in Appendix A to rebuild new schemas. In the survey, participants are asked to locate and copy types, including all the other type mentioned in its definition, into a new schema. The times (in minutes) taken for finding a type and copying it (including its dependents) were recorded for different levels of coupling. The analysis of the survey is given in Appendix G.

Survey Instructions

A: How to answer the questions

1. Type all your answers in one text file using a text editor like Notepad, Wordpad, Word, etc, and save the file under the name *YourName_SurveyResults.txt*.
2. The text file containing your answers should have your name, your class and section at the top of the first page.
3. The number of the question being answered must precede each answer.
4. The completed survey must be turned in no later than December 12

B: Searching for items in a schemas

Print out the schemas provided Appendix A, B, C, D, and E. All searching must be done manually using a printed version of the schemas provided in Appendix A, B, C, D, and E. Results that show signs of electronic searching, will receive no grade for the survey.

C: Recoding the time taken for each question

1. Time taken in answering each question must be recorded in an EXCEL worksheet.
2. The EXCEL worksheet must have your name and class at the top of the first page.
3. Each recorded time must have the question number next to it.
4. Name the EXCEL worksheet as *YourName_SurveyTime.xls*
5. Your EXCEL worksheet should be formatted as shown below:

D: Submission of results

The file containing your answers and the one containing the times must both be sent to me by email zkot2@etsu.edu

E: Grades Assignment

In order to receive a full 50-point grade, you must answer all questions.
Your results must show evidence of independent work, and also show that some level of seriousness and thought have been applied to each question.

Name		
Class-section#		
Question#	Starting Time	Ending Time

F: How long is the survey?
The bulky part of the survey consists of instructions and sample questions to guide you. Each question is carefully designed to solicit specific information about EXPRESS modules. Most of the questions should not take you more that 5minutes. If you have a question understating what is required let me know. You are not required to read and understand the EXPRESS schemas provided in the appendices in order to answer the questions.

Section 1: Importing Entities and Types

Importing. The word *import* here is used simply to mean "copy and paste". Importing a type into a new schema means copying that type *including all other types referenced in its definition*. See sample questions on next page examples below.

```
SCHEMA sample1;

TYPE label = STRING;
END_TYPE;

TYPE action_status = ENUMERATION OF
  (EXECUTED, PENDING, UNKNOWN);
END_TYPE;

TYPE age_value = INTEGER;
END_TYPE;

TYPE real_number = REAL;
END_TYPE;

TYPE integer_number = INTEGER;
END_TYPE;

TYPE char_value = STRING(1);
END_TYPE;

TYPE text = STRING;
END_TYPE;

TYPE number_select =
  SELECT (real_number, integer_number);
END_TYPE;

TYPE string_select  = SELECT
(char_value, text );
END_TYPE;

TYPE parameter_value = SELECT
 (number_select, string_select );
END_TYPE;

                ENTITY person;

   name:   STRING;
   age:    age_value;
END_ENTITY;

ENTITY measurement;
   name                : STRING;
   measure_value      : number_select;
END_ENTITY;

ENTITY address;
 city:  STRING;
 state: STRING;
 zip:   INTEGER;
END_ENTITY;

ENTITY action;
 name: STRING;
 initiator: person;
END_ENTITY;

END_SCHEMA;
```

Figure 1. Sample schema

Sample Questions for Section 1:
Assume the schema (*schema sample1*) given in Figure 1 is provided. Answer the following questions

1a) Import type label from schema *sample1* into a new schema called *sample1A.*

Explanation: In this example question, type *label* is the item to be imported. The type *label* is based on an EXPRESS base type STRING. Hence we simply copy that type into our schema. See Figure 1a.

```
SCHMEA sample1A;

TYPE label = STRING;
END_TYPE;

END_SCHEMA;
```

Figure 1a: Answer to sample question a)

1b) Import type *action_status* from schema *sample1* into a new schema called *sample1B*.

Explanation: In this example question, the item to import is *action_status.* Type *action_status* is an EXPRESS ENUMERATION. Enumeration types do not reference other types in their definitions. Hence we simply copy that type action_status; nothing else. See Figure 1b.

```
SCHMEA sample1B;

TYPE action_status = ENUMERATION OF
(EXECUTED, PENDING, UNKNOWN);
END_TYPE;

END_SCHEMA;
```

Figure 1b: Answer to sample question b)

1c) Import entity *address* from schema *sample1* into a new schema called *sample1C*.

Explanation: In this example question, entity *address* is the item to be imported. Entity *address* does not reference any user-defined type and hence we simply copy that entity; nothing else is imported with it. See Figure 1c.

```
SCHMEA sample1C;

ENTITY address;
 city:  STRING;
 state: STRING;
 zip:   INTEGER;
END_ENTITY;

END_SCHEMA;
```

Figure 1c: Answer to sample question c)

1d) Import type *number_select* from schema *sample1* into a new schema called *sample1D.*

Explanation: In this example question, type *number_select* is the item to be imported. This is a SELECT type that references two other types (*real_number* and *integer_number*) in its definition. Therefore, we need to import both *real_number* and *integer_number*.

Type *real_number* is based on EXPRESS base type REAL, so we simply copy *real_number*; nothing else is imported with it.

Type *integer_number* is also based on an EXPRESS base type INTEGER, and again we simply import type *integer_number*. See Figure 1d.

```
SCHEMA sample1D;

TYPE number_select =
  SELECT (real_number, integer_number);
END_TYPE;

TYPE real_number = REAL;
END_TYPE;

TYPE integer_number = INTEGER;
END_TYPE;

END_SCHEMA;
```

Figure 1d: Answer to sample question d)

1e) Import entity *action* from schema *sample1* into a new schema called *sample1E*.

Explanation:
In this example question, entity *action* is the item to be imported. This entity has two attributes: *name* and *initiator.* Attribute *name* is of type STRING, which is a base type. However, the attribute *initiator* references entity *person.* Entity *person* therefore needs to be imported.

In entity *person*, we also notice that type *age_value* is referenced via the attribute *age*; hence type *age_value* needs to be imported.

Type *age_value* is based on an EXPRESS base type INTEGER. We simply import the type *age_value*. See Figure 1e.

```
SCHEMA sample1E;

ENTITY action;
 name: STRING;
 initiator: person;
END_ENTITY;

        ENTITY person;

   name:   STRING;
   age:    age_number;
END_ENTITY;

TYPE age_value = INTEGER;
END_TYPE;

END_SCHEMA;
```

Figure 1e: Answer to sample question e)

1f) Import entity *measurement* from schema *sample1* into a new schema called *sample1F*.

Explanation: In this example question, the item to be imported is entity *measurement.*

Entity *measurement* references type *number_select* via attribute *measure_value.* Hence type *number_select* needs to be imported.

We also notice that *number_select* is an EXPRESS SELECT type that also references two other types in their definitions.

Importing *number_select* requires types *real_number* and *integer_number.* See Figure 1f.

```
SCHEMA sample1F;

ENTITY measurement;
   name                : STRING;
   measure_value       : number_select;
END_ENTITY;

TYPE number_select =
  SELECT (real_number, integer_number);
END_TYPE;


TYPE real_number = REAL;
END_TYPE;

TYPE integer_number = INTEGER;
END_TYPE;

END_SCHEMA;
```

Figure 1f: Answer to sample question f)

Section 1 Questions
The following questions are based on the edited version of AP203 provided in Appendix A. Please make sure you time yourself.

1) Import type *day_in_month_number* into a new schema called *Schema1A.*
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

2) Import entity product_category into a new schema called *Schema1B*
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

3) Import type *hour_in_day* into a new schema called *Schema1C*
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

4) Import entity *contract_type* into a new schema called *Schema1D*
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

5) Import type *ahead_or_behind* into a new schema called Schema1E.
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

6) Import entity *action* into new schema called Schema1F.
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

7) Import type *change_request_item* into a new schema called Schema1G.
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

8) Import entity *contract* into new schema called Schema1G
   How long did it take to complete this task?
   Start time:
   End time:
   List any other factors that made this task more difficult or easier.

9) Import entity *approval_relationship* into new schema called Schema1H.

How long did it take to complete this task?
Start time:
End time:

10) Import type *generic_definition* into a new schema called Schema1I.
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

11) Import entity *ellipse* into a new schema called Schema1J
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

12) Import entity *product_related_product_category* into a new schema1K
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

13) Import entity *chosen_action* into a new schema1L
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

Section 2: Determining Underlying Types

In EXPRESS, an attribute can have its underlying type as one of the base types, an entity type, a select type, or an enumeration type. Every attribute in a schema must have a type that determines the set of possible values that can be assigned to that attribute. There are several occasions when one needs to know the underlying type for an attribute.

Assume the following entity is given.

ENTITY test;
    attributeX: typeY;
END_ENTITY;

To determine the underlying type of *attributeX*, follow these steps:

First look for the definition of *typeY* to determine what it is.

1. If *typeY* is an ENUMERATION type
   The underlying type of *attributeX* is an ENUMERATION.

2. If *typeY* is of type SELECT
   The underlying type for *attributeX* is a SELECT.

3. If *typeY* is of type ENTITY
   The underlying type for *attributeX* is an ENTITY.

5. If *typeY* is of type defined type use the following method to get the underlying type
   Determine the underlying type for the defined type

   a) If the underlying type for the defined type is an EXPRESS base type STRING, INTEGER, NUMBER, BOOLEAN, LOGICAL
      The underlying type for *typeY* is that base type

   b) If the underlying type for the defined type is a select type then
      The underlying type for *typeY* is SELECT

   c) If the underlying type of the defined type is an enumeration type then
      The underlying type for *typeY* is ENUMERATION

   d) If the underlying type for the defined type is another defined type then
      Repeat steps a) to d).

Sample questions for Section 2:
The following examples questions are based on the schema in Figure 1.

2a) Determine the underlying type for attribute *status* in entity person? What are the possible values that can be assigned to attribute *status*?

Answer:
Underlying type: ENUMERATION

2b) Entity *measure* has an attribute called *measure_value*. What is the type name for attribute *measure_value?* Determine the underlying type for the attribute *measure_value.*

Answer:
Type name for attribute meaure_value is number_select.
Underlying type: SELECT

2c) Determine the underlying type for attribute *initiator* in entity *action?*

Answer:
Underlying type for attribute initiator is ENTITY.

2d) Determine the underlying type attribute for *age_number* in entity *person*?

Answer: INTEGER
The attribute age_number is a defined type, which is based on an EXPRESS base type INTEGER.

Section 2 Questions
The following questions are based on the edited version of AP203 provided in Appendix A. Perform each task and record the time taken. Please make sure to time yourself.

14) Entity *face_bound* has an attribute called *bound*. What is the type of the attribute *bound*? What is the underlying type?
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

15) Entity *si_unit* has an attribute called *prefix*. What is the type name and underlying type for the attribute *prefix*?
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

16) In entity *coordinated_universal_time_offset*, there is an attribute called *sense*, what is the underlying type for the attribute *sense*?
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

17) Entity *measure_with_unit* has an attribute called *value_component*. What is the type name and underlying type of attribute *value_component*?
How long did it take to complete this task:
Start time:
End time:
List any other factors that made this task more difficult or easier.

18) Entity *geometric_representation_context* has an attribute *coordinate_space*, what is the type name and the underlying type of the attribute *coordinate_space*?
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

19) Entity *b_spline_curve_with_knots* has an attribute *knot_spec*, what is the type name and the underlying type of the attribute *knot_spec*?
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

20) Entity *circle* has an attribute called *raduis*. What is the type name and the underlying type for the attribute *raduis*?
How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

Section 3: Complex Domains
The domain of a type is the set of values that the type is limited to. The following questions ask for the domain of certain types. To determine the domain of a type follow the following algorithm.

Assume the following entity is given.

ENTITY test;
attributeX: typeY;
END_ENTITY;

1. If *typeY* is an EXPRESS ENUMERATION
   The domain of *typeY* is the set of values mentioned in the enumeration list.
   (See Sample question 3a)

2. If *typeY* is a defined type
   The domain of *typeY* is the domain of the type that the defined type is based on
   (See Sample question 3b)

4. If *typeY* is of type ENTITY (for the purpose of this survey)
   The domain of *typeY* is one of the following:
   e) All subtypes of entity *typeY*
   f) The entity *typeY* itself (except where *typeY* is an abstract supertype)
   (See Sample question 3c)

6. If *typeY* is an EXPRESS SELECT (note a Select type has a list of types in its select list)
   For each type mentioned in the select list, determine the domain using steps 1,2,3,4.
   The domain of typeY is the sum of the domains of all types in the select list.
   (See Sample question 3d)

Sample Question for Section 3

The following sample questions are based on the schema in Figure 1.

3a) (Domain of an ENUMERATION type) What is the domain of type *action_status*?

Answer:
Domain of type action_status ={EXECUTED, PENDING, UNKNOWN }

3b) (Domain for a Defined type) What is the domain for type label?
Answer:
Domain of type label is domain of base type STRING.

3c) (Domain of an entity type) Attribute *initiator* in entity *action* has a type *person* where *person* is an entity type. What are the possible types that can be assigned to the attribute *initiator*? In other words what is the domain for type *person?*

Answer:
Attribute initiator is of type person, which can be one of the following:
{ student | professor| studentprofessor } which is same as the domain of type person.

3d) (Domain of a SELECT type) What is the domain of type *parameter_value?*

Answer:
Domain of type parameter_value ={ number_select, string_select }

Domain of type number_select={real_number, intege_number }

Domain of type string_select = {char_value, text}

Complete domain of type parameter_value ={ real_number, integer_number, char_value , text } which same as {REAL, INTEGER, STRING(1), STRING}

Section 3 Questions
The following questions are based on the edited version of AP203 provided in Appendix A. Perform each task and record the time taken. Please make sure you time yourself.

21) Entity *b_spline_curve_with_knots* has an attribute called *knot_spec*. What is the type name fro the attribute *knot_spec*?
    How long did it take to complete this task?
    Start time:
    End time:
    List any other factors that made this task more difficult or easier.

22) Entity *assembly_component_usage_substitute* has an attribute called *base*. List the possible types that can be assigned to the attribute *base.* (in other words, what are the possible types that the attribute *base* can assume) ?
    How long did it take to complete this task?
    Start time:
    End time:
    List any other factors that made this task more difficult or easier.

23) List all the types that make up the complete domain for type *formal_approval?*
    How long did it take to complete this task?
    Start time:
    End time:
    List any other factors that made this task more difficult or easier.

24) Entity *valid_reference_source* has an entity called *source*. List the possible types that can be assigned to the attribute *source* can assume (i.e. in other words, what is the domain of type *approved_source_of_reference*)?
    How long did it take to complete this task?
    Start time:
    End time:
    List any other factors that made this task more difficult or easier.

Section 4: Inheritance Hierarchies

Root of an inheritance tree. The root of an inheritance tree (hierarchy) is the uppermost entity without any supertype. The following questions ask you to determine the supertypes and subtypes as well as the roots in certain inheritance hierarchies. The example below is provided to guide you.

Sample Questions for Section 4

a) Entity *undergrad_student* is in a simple inheritance hierarchy. What is the root of this inheritance hierarchy?

   Answer: person

b) What is the direct supertype of entity *undergrad_student?*

   Answer: student

```
SCHEMA sample3;

          ENTITY person
       ABSTRACT SUPERTYPE OF( ONEOF
       (male, female)
       ANDOR
       (student, professor) );

   name:   STRING;
   age:    age_number;
END_ENTITY;

            ENTITY male
 SUBTYPE( person );
END_ENTITY;

          ENTITY female
 SUBTYPE( person );
END_ENTITY;

          ENTITY student
 SUBTYPE( person );
END_ENTITY;

       ENTITY grad_student
 SUBTYPE( student );
END_ENTITY;

    ENTITY undergrad_student
 SUBTYPE( student );
```

Figure 3: Sample schema for Section 4 questions

Section 4 Questions

The following questions are based on the edited version of AP203 provided in Appendix A. Perform each task and record the time taken. Please make sure you time yourself.

24) Entity *calendar_date* is part of a simple inheritance hierarchy. What is the root of this inheritance hierarchy?

How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

25) Entity face_bound is also part of an inheritance hierarchy. What is the root of this inheritance hierarchy?

How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

26) Find the root of the inheritance hierarchy that entity *conic* is part of.

How long did it take to complete this task?
Start time:
End time:
List any other factors that made this task more difficult or easier.

APPENDIX G

SURVEY RESULTS—TIME FOR REUSING EXPRESS MODULES

This Appendix presents the results obtained from the survey. Each column in the table shows the type and level of coupling of the type being searched for in each question. The rows give the reported times. For instance, for DAC_ENT level 0, two questions were asked (Question 2 and, Question 4). The times (in minutes) recorded for these questions were added and averaged. A summary of the analysis of this data is also given in Chapter 5. (See Appendix F for survey questions). All times are in minutes recorded to one decimal place.

DAC_ENT

Time (min) for Level 0 Survey Question 2	Survey Question 4	Time (min) Level 1 Survey Question 6	Survey Question 14	Time (min) Level 2 Survey Question 8	Survey Question 9	Time (min) Level 3 Survey Question 12	Survey Question 13
2.0	1.0	4.0	1.0	5.0	1.0	8.0	5.0
4.0	5.0	5.0	8.0	3.0	9.0	6.0	3.0
1.3	0.8	0.9	4.0	2.0	6.0	2.2	1.8
3.0	1.0	2.0	15.0	4.0	1.0	8.0	3.0
7.0	4.0	5.0	4.0	9.0	5.0	10.0	10.0
4.0	1.0	3.0	2.5	6.0	6.0	5.0	2.0
1.0	0.5	2.0	2.0	1.5	2.2	2.5	3.6
2.0	1.0	1.0	9.0	2.0	2.0	2.0	1.0
5.0	3.0	1.4	4.0	4.0	3.0	8.0	4.0
6.0	3.0	4.0	6.0	6.0	1.0	15.0	9.0
2.0	2.0	4.0	2.0	4.0	1.0	10.0	6.0
2.0	2.0	4.0	2.0	6.0	2.0	11.0	9.0
2.3	1.8	3.0	4.0	5.5	1.8	20.0	5.0
3.0	3.0	3.5	5.0	3.0	10.0	2.0	2.0
1.0	2.0	3.0	1.0	3.0	5.0	4.0	3.0
1.7	2.8	3.0	3.5	2.3	1.6	2.5	2.8
2.0	1.2	2.5	2.0	3.2	2.0	1.5	2.0
8.5	1.8	2.3	2.0	13.0	9.0	10.0	2.0
2.0	1.0	2.0	2.0	4.0	5.0	2.0	5.0
3.0	4.0	5.0	2.0	7.0	3.0	14.0	5.0
6.0	3.0	3.0	8.0	4.0	9.0	3.0	3.0
6.0	2.0	3.0	4.0	7.0	3.0	12.0	5.0
5.0	2.0	7.0	4.0	5.0	2.0	24.0	8.0
1.8	1.5	3.3	2.3	3.5	1.5	11.6	4.5
3.0	2.0	5.0	2.0	6.0	1.0	5.0	4.0
4.0	6.0	6.0	10.0	8.0	10.0	6.0	2.0
5.0	2.0	4.0	15.0	4.0	4.0	9.0	8.0
4.3	1.3	3.0	5.0	4.2	3.0	9.0	3.5
3.0	2.0	5.0	2.0	4.0	4.0	10.0	7.0
1.0	3.0	2.0	1.0	4.0	2.0	8.0	4.0
5.0	3.0	6.0	9.0	4.0	3.0	6.0	4.0
Sum	176.4		251.2		266.3		384.5
Mean	2.85		4.05		4.30		6.20

Inheritance

	Time (min) for Level 1	Time (min) for Level 2	Time (min) for level 3
	Survey Question 24	Survey Question 25	Survey Question 26
	2.0	2.0	2.0
	4.0	3.0	2.0
	0.6	1.0	1.0
	1.0	1.0	1.0
	3.0	5.0	8.0
	3.0	1.0	1.0
	2.0	2.0	1.5
	1.0	1.0	1.0
	1.0	2.0	4.0
	1.0	2.0	5.0
	1.0	1.0	1.0
	2.0	3.0	2.0
	1.2	2.3	2.2
	6.0	8.0	9.0
	2.5	4.0	1.0
	2.0	1.0	4.0
	1.2	4.0	3.0
	9.0	6.0	11.0
	2.0	2.0	2.0
	3.0	1.0	3.0
	4.0	2.0	5.0
	2.0	2.0	3.0
	2.0	4.0	4.0
	1.2	1.0	1.2
	1.0	2.0	2.0
	4.0	3.0	3.0
	4.0	9.0	2.0
	1.3	2.5	1.3
	3.0	3.0	5.0
	1.0	2.0	1.0
	2.0	1.0	1.0
Sum	74.0	83.8	93.2
Mean	2.39	2.70	3.00

DAC_SEL

	Time (min) for Level 1		Time (min) for Level 2	Time (min) for Level 3
	Survey Question 7	Survey Question 17	Survey Question 10	Survey Question 24
	4.0	2.0	4.0	3.0
	2.0	5.0	6.0	5.0
	3.5	0.8	2.6	1.5
	3.0	1.0	4.0	5.0
	4.0	3.0	9.0	1.2
	9.0	2.0	15.0	4.0
	2.0	1.3	2.0	1.0
	1.0	1.0	1.0	4.0
	3.0	2.0	5.0	11.0
	5.0	5.0	6.0	1.0
	4.0	2.0	6.0	3.0
	7.0	3.0	9.0	15.0
	4.8	2.0	6.0	6.0
	2.0	3.0	5.0	4.0
	2.0	7.0	2.0	4.0
	5.0	2.0	4.0	5.3
	3.0	2.5	1.5	17.0
	4.0	4.0	1.5	2.0
	1.0	2.0	1.0	1.3
	6.0	2.0	7.0	2.0
	3.0	2.0	2.0	5.0
	5.0	4.0	5.0	4.0
	8.0	2.0	7.0	12.0
	3.3	1.3	4.5	2.0
	3.0	2.0	5.0	12.0
	3.0	6.0	6.0	4.0
	3.0	2.0	4.8	6.0
	4.5	1.8	6.0	4.0
	7.0	2.0	5.0	3.0
	4.0	2.0	5.0	2.0
	3.0	2.0	2.0	4.0
Sum	201.7		149.9	154.3
Mean	3.35		4.84	4.98

DAC_DEF

	Time (min) for Level 1	Time (min) for Level 2	Time (min) for Level 3
	Survey Question 1	Survey Question 18	Survey Question 20
	3.0	2.0	1.0
	5.0	5.0	6.0
	1.3	1.0	1.3
	2.0	1.0	2.0
	4.0	4.0	6.0
	1.0	3.0	2.0
	1.3	1.0	1.5
	2.0	3.0	2.0
	3.0	2.0	3.0
	7.0	5.0	4.0
	2.0	2.0	5.0
	4.0	2.0	6.0
	2.5	3.5	2.0
	2.0	3.0	5.0
	3.0	6.0	3.0
	2.0	3.0	1.0
	2.5	4.0	4.0
	1.0	2.0	1.8
	1.8	2.0	1.0
	3.0	8.0	6.0
	6.0	1.0	1.0
	2.0	1.0	1.8
	3.0	2.0	4.0
	2.0	3.0	2.0
	2.0	2.0	3.0
	2.0	3.0	4.0
	3.0	1.0	5.0
	4.0	2.0	5.0
	2.0	3.0	3.0
	3.0	2.0	2.0
	2.0	3.0	1.0
Sum	84.4	85.5	95.4
Mean	2.72	2.76	3.08

VITA

KWAKU OWUSU-TIEKU

Personal Data:	Date of Birth: March 14, 1972 Place of Birth: Adomfe, Ashanti Akim, Ghana
Education:	Port Moresby International High School, Port Moresby, PNG East Tennessee State University, Johnson City, Tennessee, Computer Science, BS, 1998 East Tennessee State University, Johnson City, Tennessee, Information Science, MS, 2001
Professional Experience:	Graduate Assistant, East Tennessee State University, Johnson City, Tennessee, 1998-2000 Software Designer/Programmer, Department of Computer & Information Sciences, East Tennessee State University, Johnson City, Tennessee, 2000-2001 Software Engineer, Sprint PCS, Nashville, Tennessee, 2001