Special Report
CMU/SEI-92-SR-3

Joint IntegratedAvionics Working Group (JIAWG) Object-Oriented Domain Analysis Method (JODA)

Version 3.1

Robert Holibaugh

November 1993

Special Report

CMU/SEI-92-SR-3

November 1993

Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis Method (JODA)

Version 3.1

Robert Holibaugh

Joint Integrated Avionics Working Group

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.   Phone: (703) 274-7633.

Table of Contents

List of Figures

Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis (JODA) Method

Abstract: The Joint integrated Avionics Working Group (JIAWG) Reuse Subcommittee has initiatives in several areas to demonstrate that reuse can effectively support the JIAWG programs, and the creation of reusable assets is an essential element of reuse. Domain analysis is the process that identifies what is reusable, how it can be structured, and how it can be used. This report describes a method for domain analysis that is based on Coad and Yourdon's *Object Oriented Analysis.* This method, the JIAWG Object-Oriented Domain Analysis (JODA), includes several enhancements to the method of Coad and Yourdon and produces a domain model to support asset creation and reuse.

1 Introduction

The Joint Integrated Avionics Working Group (JIAWG) is a Tri-Service effort mandated by Congress to exploit maturing technology to realize the economic, supportability, and interoperability advantages of common avionics hardware and software. The JIAWG Reuse Subcommittee, which is part of the Software Task Group, has developed initiatives in several areas to demonstrate that software reuse can effectively support the JIAWG programs [A-12, the Air Force's Advanced Tactical Fighter (ATF), the Army's Light Helicopter (LH), the Air Force's Advanced Tactical Aircraft (AF ATA), the Navy's ATF (NATF), and A-12 Preplanned Product Improvement (P^3I)]. The Reuse Subcommittee initiatives are: domain analysis, contract incentives, standards to support reuse, and reuse libraries.

The JIAWG Reuse Subcommittee's domain analysis group has been chartered with:

- selecting, defining, documenting, and refining a domain analysis method,
- performing an example avionics domain analysis and documenting the results, and
- documenting the lessons learned from the domain analysis effort.

1.1 Purpose

The purpose of this report is to document the JIAWG Object-Oriented Domain Analysis (JODA (pronounced as if written "yoda")) method that could be used by the JIAWG programs for domain analysis. This method has already been used by the JIAWG Reuse Subcommittee's domain analysis group to analyze an avionics domain, stores management. This method and the results of the stores management analysis are intended as examples for the JIAWG Systems Program Offices (SPO) and contractors. The Reuse Subcommittee domain analysis effort is also intended to demonstrate the effectiveness of domain analysis technology for avionics software.

The approach presented in this document is only one of several views of a reuse life cycle. (For other examples, see references 14, 25, and 29.) This domain analysis method is based on the object-oriented analysis (OOA) techniques and notation defined in *Object-Oriented Analysis* by Coad and Yourdon [7], which we will refer to as CYOOA. In this document, CYOOA notation has been broadened to include: performance issues, scenarios for controlling the objects, and rationales for how, when, where, and why to use the objects when building systems. These additions were necessary since CYOOA was originally intended for requirements analysis in the construction of a single system. CYOOA notation was enhanced to validate and demonstrate the effectiveness of object-oriented techniques to support domain analysis.

We believe that software objects are more understandable, more adaptable, and less likely to change than functions. Tactical aircraft will always carry bombs, so a bomb object will always be part of a Stores Management System. The functions needed to manage and deliver a bomb will change over time, but the software changes are restricted to the bomb object, making it easier to adapt that object [5, 16]. Finally, the bomb object's functions are limited to the state defined by the object thus making the object easier to understand and change, since the maintainer need only understand the bomb object and not all of Stores Management. For these reasons, object-oriented techniques have been chosen instead of a functional approach. Our domain analysis goal is to represent the requirements with OOA notation in a domain model that can be used to produce object-oriented requirements, designs, code, and tests.

The office building elevator systems (OBE) example was specifically chosen to help illustrate the domain analysis method and concepts. (The OBE problem statement is given in Appendix B.1.) The OBE system that provides safe and equitable service for its passengers for some number of elevators that service several floors was chosen because:

- the OBE is well understood by a large audience,
- the OBE illustrates the concepts of the method,
- the OBE illustrates the breadth and depth of even simple domains, and
- a clear, concise problem description was readily available.

1.2 Goals of Domain Analysis

The goal of domain analysis is to define a domain model that can be used to produce reusable software objects (RSO), especially reusable requirements. The information in the domain model is collected using CYOOA techniques. Domain analysis is part of the domain engineering process that uses the domain model to define a reusable software architecture, to design reusable code (a more detailed level of design than architecture), and to define the structure of the domain. The domain structure is the organization of the parts as dominated by the general character of the whole [24]. The domain structure is defined by the CYOOA diagrams that define the composition of the domain through whole-part diagrams and the variation in objects using generalization-specialization (gen-spec) diagrams. The objects are defined by their service and their attributes, while domain structure is defined though whole-part concepts, variation in objects, services, attributes, and concepts, and the relationships between these objects and concepts.

1.3 Background

Gilroy, et. al., conclude in their research that, “Domain analysis, when done right, is a significant undertaking yet produces a significant benefit [13].” At a high level, domain analysis is a combination of reverse engineering, knowledge extraction, knowledge representation, requirements forecasting, and technology forecasting. In domain analysis, the essential concepts are extracted, represented, and adapted for reuse. Knowledge extraction and representation are used to establish a domain framework, while reverse engineering fills in the details and validates the framework. Technology forecasting and requirements forecasting techniques are used to ensure the results remain viable while the investment in domain analysis is recovered. Recent literature on reuse indicates that domain analysis is one of the first activities that should be performed during the engineering of reusable software [21, 29]. Organizations that have conducted domain analysis prior to creating reusable software components have shown greater success in reusability [18, 20]. Reusable components that are constructed from the results of domain analysis capture the essential concepts required in that domain; thus, developers find them easier to include in new systems [28].

After reviewing the literature on domain analysis, the author recognized that the information that is commonly collected by domain analysis methods is essentially the same as that defined by the CYOOA notation. For two additional reasons, CYOOA was selected and enhanced to support our domain analysis needs. First, other domain analysis efforts use CYOOA notation in their domain analysis products. Second, selecting a commercial analysis technique makes training, tools, and consulting support readily available. There was one major difference between CYOOA and other domain analysis representation techniques. The information that was being represented and organized using CYOOA notation was accessible, understandable, and concise. Even though there is no consensus on what results to represent in domain analysis, the core set of our needs is met by CYOOA.

After further examination of the CYOOA notation and method, another domain analysis effort was identified that uses CYOOA for its domain model [29]. The Software Productivity Consortium (SPC) has made minor additions to the notation; for example, SPC has added a textual discussion of the variation in classes, a textual discussion of the occurrence of the number of objects, and a discussion of performance requirements on the overall system. Since CYOOA was being used by other reuse efforts and had commercial acceptance, training, and tools, CYOOA techniques were chosen for extracting, organizing, and representing the domain model.

In typical Department of Defense (DoD) software development, only the requirements, designs, code, and test materials are usually recorded, and if domain knowledge is recorded, it is never delivered. The value of the domain knowledge that is acquired by the developer is frequently not even recognized, and therefore, it is not available for post deployment software support (PDSS). Since the JIAWG aircraft may be in the DoD inventory for as long as twenty years, PDSS which is also concerned with change and variation is a major concern of the JIAWG Systems Program Offices (SPO). PDSS includes two activities: first, correcting problems with the system, and second, making necessary enhancements. To locate and correct systems problems, the maintainer must understand the system requirements, design, and code. The organization of information collected during domain analysis supports identifying, locating, and correcting the problem, since the domain knowledge helps the maintainer understand the what, how, when, where, and why for the system data and services. The second major PDSS activity includes making changes or enhancements to the system.

The domain model contains the information that is necessary (but seldom available) for making changes or enhancements to an existing system. The domain model includes the rationale for all domain services, attributes, and objects. In fact, a desired PDSS change or enhancement may already be in the model, and its inclusion in the system may be relatively simple. Domain analysis supports post deployment software support as well as reuse, because it captures and anticipates change.

1.4 Relationship of Object-Oriented Analysis to JODA

Object-oriented analysis techniques are used to define the structure and capture requirements, but Coad and Yourdon's Object-Oriented Analysis (CYOOA) is not the same as the JIAWG Object-Oriented Domain Analysis (JODA). JODA and CYOOA differ in both notation and process: JODA has enhanced the CYOOA notation and process. The specific areas where JODA and CYOOA differ are as follows:

- The problem statement's scope is fixed before starting CYOOA, but it is not fixed in domain analysis.
- CYOOA notation addresses a single system while JODA notation addresses a family of systems.
- CYOOA notation does not include scenarios definitions that have been include in JODA to define the use of the domain's visible services.
- An abstraction activity that is not in CYOOA has been included in the analysis phase of JODA.
- A scenario definition and walk-through activity has been added to the modeling phase of JODA.

The problem statement is fixed at the beginning of CYOOA while it changes after the beginning of the JODA process. CYOOA is a software requirements analysis technique [7,12] that has the problem defined by System Design [8, 12] since CYOOA is intended to support the software engineering process. Domain analysis receives input from the Business and Methodology Planning steps [17], but these steps do not define the domain or bound the problem. Domain analysis must define the domain in addition to identifying, locating, and collecting source material.

The second difference between CYOOA and JODA concerns the scope of the problem. CYOOA is intended to address the analysis of a single problem and does not have all the notation and techniques required for defining variation in the domain. These deficiencies do not prevent the use of CYOOA; rather, by adding to the notation, CYOOA can be enhanced to specify a domain instead of a single problem. Since CYOOA allows for the inclusion of additional notation, this can be done easily. For example, CYOOA does not contain a rationale for describing how, when, where, and why to select options, services, and instances. (Rationale is necessary when the reuser must select between options in the domain.) Also, CYOOA notation does not contain a mechanism for defining requirements which range over multiple objects and a mechanism for specifying real-time performance parameters. Additions have been defined in JODA to specify these requirements.

CYOOA does not have notation for defining scenarios that are a series of object services that produce a larger capability such as the stores management services that a pilot uses to release a bomb. Scenarios and their description are essential to domain and software requirements analysis. While a user's manual will describe scenarios that use the system's services, there is no user's manual for domain analysis. Scenarios have been added to CYOOA to identify high-level capabilities in the domain. Notation has also been added to define the rationale for each whole-part relationship, instance connection, and gen-spec relationship.

JODA and CYOOA also differ in analysis technique. The JODA analysis technique is an extension of CYOOA. The CYOOA analysis defines five activities, but the order of the activities is not fixed. (See Appendix C.) JODA does not prescribe an ordering, but it adds two additional activities: (1) identify and walk-through scenarios, and (2) abstract and group objects. CYOOA does not emphasize identifying scenarios or abstracting objects across systems.

The domain analysis team must identify, document, and simulate scenarios to validate the objects and their relationships. One reason for including scenarios is that high-level services like delivering a bomb are a combination of other domain services. These scenarios are important for a user to obtain a gestalt for the domain. The simulation of scenarios is a basic technique used by experts to develop systems [1]. In JODA, these scenarios are explicitly identified, defined, and simulated by the analysts. The identification and definition helps organize and clarify the domain requirements. The simulation validates the classes, attributes, and services that have been specified. Scenarios help the domain analyst and the users to understand and apply the results of domain analysis.

The process of abstracting domain analysis results is similar to but not the same as defining gen-spec structure. The abstraction process includes defining additional gen-spec structure, but it also includes identifying subjects (as in Figure A-5). This subject diagram provides an overview of the objects and their relationships for OBE systems. These abstractions are part of the CYOOA notation, but abstraction is not applied so that the results cover several systems. The merging of objects and services from similar systems into an abstract representation supports an integrated view of the domain. This integrated view is the goal of the abstract and group objects activity of domain modeling. This goal is important because the user of the domain model can analyze the subjects and their associated objects on at a time. This helps prevent confusion and reduces that amount of detail to an acceptable level. Another abstraction example is the definition of Doors in Appendix B.9. Three door types (elevator lobby, elevator, and elevator rear doors) were defined during the initial pass through the domain. These door had similar services, but the abstraction was not identified until after the CYOOA analysis was complete. The classes Services Button (as was the style in CYOOA, all objects from the OBE example in this document will be capitalized in the text to identify them) in Appendix B.5 and Status Indicator in Appendix B.7 were also identified during the emphasis on abstraction. These abstractions have been useful for describing the domain especially at the high level.

Two activities have been added to CYOOA analysis: identifying and walking through domain scenarios and abstracting and grouping objects. These additions enhance CYOOA to specify a family of systems, and they can be interleaved with other domain analysis activities.

1.5 Relationship of the JIAWG Method to the Other Domain Analysis Methods

The JIAWG domain analysis method is similar in many respects to the SPC approach [29], but it also differs in some distinct ways. Before examining the similarities and differences, the SPC products and process are listed. The SPC approach to domain analysis has four steps, and each step creates deliverable products. The SPC steps and their products are:

- Domain Description produces domain definition and conceptual taxonomy,
- Domain Qualification produces a feasibility analysis,
- Knowledge Base Creation produces a domain knowledge base, and
- Canonical Requirements Development develops the reusable statement of domain requirements.

Three steps in the SPC approach have a corresponding phase in JODA. The domain description step in the SPC approach is very similar to the JODA domain definition phase. Both efforts produce a domain definition that controls the scope of the later analysis effort. The SPC's Knowledge Base Creation step identifies, locates, and gathers references and source material. This step is similar to the Domain Preparation phase in JODA. The SPC's last phase is similar to the Domain Modeling phase of JODA. The goal of these activities in both methods is to produce reusable domain requirements. Both the SPC approach and JODA use CYOOA to define the domain model. The SPC Canonical Requirements Development does not address the format of reusable requirements. For the JIAWG, the requirements must be compatible with DoD-STD 2167A DIDs. The model/requirements transformation activity transforms the CYOOA notation into SRS format. Before we examine the differences in the two methods, the use of CYOOA notation is examined.

JODA and the SPC approach use the CYOOA to define the requirements, but both methods have made enhancements to the notation. The SPC has identified three additions to the

CYOOA specifications; they are:

- a textual description of object variation,
- a textual description of the occurrence of objects, and
- performance requirements for objects and their services.

These three additions are used by SPC to document the details of all objects. JODA does not include a textual description of object variation or occurrence, but it does include performance requirements. The JIAWG method has also included notation for documenting scenarios and the rationale and guidelines for the how, when, where, and why for each variation. The JODA rationale for each variation is related to SPC's textual variation. For each variation, JODA gives the rationale for when and where that variation should be used. (The scenarios that JODA has added are described in the previous section.)

The CYOOA analysis and notation techniques are core concepts to each approach, but the SPC and JIAWG domain analysis methods differ in the following ways. The SPC method includes an economic analysis while the JIAWG method does not; the SPC approach does not address converting the requirements to DoD standard format. From a reuse-based software life cycle perspective [17], economic analysis is performed during Business Planning. JODA expects the general domain area to have been selected, and it refines that definition based on the domain and the time and resources available. The differences between the two methods can be attributed to the perspectives of the two authors. JODA has focused on the analysis activity and the use of the results by JIAWG SPOs and contractors, and JODA considers the economic analysis to be part of a separate activity. SPC emphasizes the importance economic aspects and domain analysis. SPC's approach is designed to support the Synthesis Method [6], while JODA does not endorse a particular applications engineering method.

1.6 Organization of the Report

This document assumes that the reader is familiar with the notation and process presented in CYOOA, and the report describes a domain analysis method based on CYOOA techniques for analysis and representation. The domain analysis products are defined using the notation of CYOOA, and most of the examples have been created using CYOOA. If one reads this document without an understanding of CYOOA, then the description of the domain analysis products will be confusing. Furthermore, since the definition of the products drive the process, the reasons for some of the process steps will be unclear.

This report provides the reader with an overview of domain analysis and a detailed description of the products and the process. Chapter 1 provides background information on domain analysis CYOOA while Chapter 2 sets the context for domain analysis. Chapter 3 is an overview of the domain analysis method. Chapter 4 describes the domain analysis products in detail, while Chapters 5 through 7 discuss the three phases of domain analysis. Chapter 8 considers the transition to domain implementation and the relationship of the domain analysis and DoD MIL-STD-2167A. The Appendices contain examples of the application of object-oriented analysis to elevator systems and a brief description of the activities and notation for documenting the domain model.

2 Context for Domain Analysis

In this chapter, we define the context for domain analysis. Figure 2-1 defines a high-level context for domain analysis. Figure 2-1 also shows how domain engineering can be included in a software life cycle. Business and Methodology Planning are necessary parts of both Applications Engineering and Domain Engineering, while Domain Engineering is not a necessary part of Applications Engineering. Feedback from Applications Engineering is shown to highlight its importance. Each application which is reuse-based acquires new information that needs to be entered into the domain model. The new information may require the creation, modification, or deletion of RSOs. Without this feedback, the maximum return on the reuse investment cannot be achieved, and the domain model and RSOs will not be current or viable. Below we briefly describe each activity in Figure 2-1; the relationship of Domain Analysis to Domain Engineering and Applications Engineering is more fully examined in the following sections. The activities in a Reuse-Based Life Cycle are: Business Planning, Methodology Planning, Domain Engineering, and Applications Engineering.

Business Planning

This goal of this activity is to identify and select high-level domains that will be considered for domain analysis. The criteria for identifying a domain are: is the domain is well understood, is the technology predictable, and is domain expertise available to support domain engineering? The risk of performing domain analysis must also be considered, since domain analysis technology is still not mature and requires a major investment [23]. Finally, there must be an opportunity to recover the investment and show a return on domain engineering. The candidates for domain analysis need not be rigorously defined: they can be high-level domains like command, control, and communications or avionics.

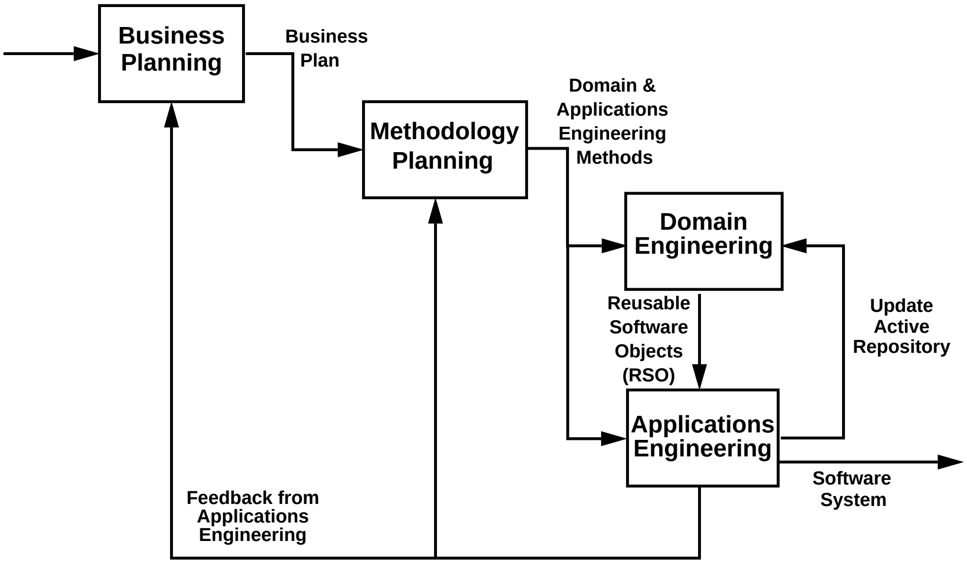


Figure 2-1 Reuse Based Software Development

Methodology Planning

The goal of this activity is to define a set of methods for domain engineering that is compatible with the methods for applications engineering. If the methods are not compatible, then the domain knowledge and software objects may not be reusable. CYOOA has been selected as the basis for the JIAWG domain analysis, and the applications engineering method must be compatible. Objects can be reused when applications engineering is based on a functional approach. The type of domain will affect the method to be used. If the domain has hard real-time requirements, then the domain engineering and applications methods must be able to represent hard-real time requirements, design, tests, and implementation for domain engineering. The corresponding method in applications engineering must produce software deliverables which support hard real-time systems. For example, code generators may not be able to meet this hard real-time need.

Domain Engineering

Figure 2-2 provides more detailed information about the Domain Engineering activity including the flow of specific data within Domain Engineering and what data are input to Applications (Software) Engineering. Domain Engineering acquires and represents information that is used to create RSOs that are reused during Applications Engineering. Note that Figure 2-1 shows the feedback from applications (software) engineering to Domain Engineering. This feedback is necessary to maintain the viability of the RSOs. The goal of each domain engineering activity is discussed in Section 2.1.

Applications Engineering

The specific data that is used during Applications Engineering is identified in Figure 2-2. Each phase of the DoD-STD 2167A Life Cycle uses the results of Domain Engineering. The domain model is used during Systems Requirements Analysis to produce a SRS. Note that Figure 2-1 shows the feedback from Applications (software) Engineering to Domain Engineering. This feedback only occurs when the software engineers and management actively support reuse during Applications Engineering. This feedback is necessary to evolve the domain model and RSOs and to support the evolution of reuse into a mature strategy for Applications Engineering.

2.1 Domain Engineering

The goal of domain engineering is to capture, organize and represent the domain from which RSOs are produced to support implementing any member of the domain (a family of systems). The activities in domain engineering are similar to software engineering, but there are two significant differences. First, domain engineering attempts to define a software solution (a family of systems) for a large problem space, while software engineering constructs only one solution (a single system) that is usually a subset of the larger problem space. This difference between domain engineering and software engineering is analogous to the differences in providing a solution to the Quadratic Equation, 2Ax + Bx + C = 0, as opposed to finding the solution to one specific equation such as $2X^2 - 230X - 45439$. Variation in the problem space makes one solution more general than another.

To illustrate the impact of variation on a domain, consider the following. In high school algebra, we learned several techniques for solving quadratic equations. The ability to factor quadratic equations (common attribute) is enhanced by knowing when factoring is possible. By examining the general solution, we recognize that $B^2- 4AC > 0$ is necessary for factoring to be feasible. Since A can always be made positive, if $C < 0$, then $B^2-4AC > 0$, and the equation has real solutions. Thus, one rationale for selecting factoring to solve a quadratic equation is that C must be less than 0. The factoring solution is enhanced by identifying and analyzing variations in the domain and the rationale for applying them. One major difference between domain engineering and software engineering is that domain engineering provides a solution for a family of systems, and variation in the problem space defines the family.

The second major difference is that software engineering does not attempt to represent and deliver the domain knowledge that has been acquired. During software development, the analyst acquires important domain knowledge which is rarely recorded and maintained. Domain engineering specifically gathers, represents, and maintains that knowledge since it is necessary for reuse. In domain analysis, the problem space for which we seek a general solution, is analyzed, defined, and specified. Rationale that defines how, when, where, and why for each variation in the domain is included in the model. Once this has been done, general solutions are identified, represented, and engineered for use. Domain engineering yields two classes of products:

- A representation of the domain structure, requirements, architectures, concepts, foundations, and expert opinions, and
- Reusable Software Objects (RSO) such as requirements, designs, algorithms, code, and tests.

The process of domain engineering is composed of three activities: Domain Analysis, Domain Implementation, and Active Repository. Figure 2-2 defines the data flow. These activities are described below.

2.1.1 Domain Analysis

The goal of domain analysis is to define the domain structure and requirements and capture them in a domain model. To adequately understand the domain, existing systems must be analyzed to identify the domain's traditional software requirements. Domain experts are interviewed to define high-level domain abstractions and to verify the information obtained from the analysis of existing systems. Future trends in requirements and domain technology must be identified to ensure that the results remain viable so that a return on the domain analysis investment is obtained. The information derived from domain analysis is organized into a domain model that is used during Applications Engineering to produce DoD standard requirements. These requirements are used in Applications Engineering to produce a Software Requirements Specification (SRS). The domain model defines the domain for reuse, but the model may only be complete at an abstract level.

The inclusion of all requirements in the domain model is not effective. The model should not contain requirements that are obsolete, one-of-a-kind, or arbitrarily allocated. The domain model should be complete at an abstract level. If the model identifies all the detailed requirements, it will become rigid and difficult to understand. When the team members analyze existing systems, they will include an abstraction of the objects, services, attributes, and relationships in the model. The model will directly reflect one existing system when that system represents all the others. Normally, the model will not include obsolete requirements. In fact, the model may not include current requirements if future trends will make those requirements appear obsolete. The model will not contain unique requirements if the requirements are not normally included in the domain. Finally, the model does not contain capabilities that have been included in the domain, but could have been allocated to other domains without degrading performance or capability.

The analysis team will define the domain, and review questions and issues with the domain experts. The team and the domain experts should reach consensus on determining the level of abstraction and defining the domain. Abstract requirements which captures the essence of several requirements are preferred over more detailed requirements. If the domain analysis team determines that a requirement is unique, obsolete, or should not be included, and the domain experts concur, then that requirement will be excluded.

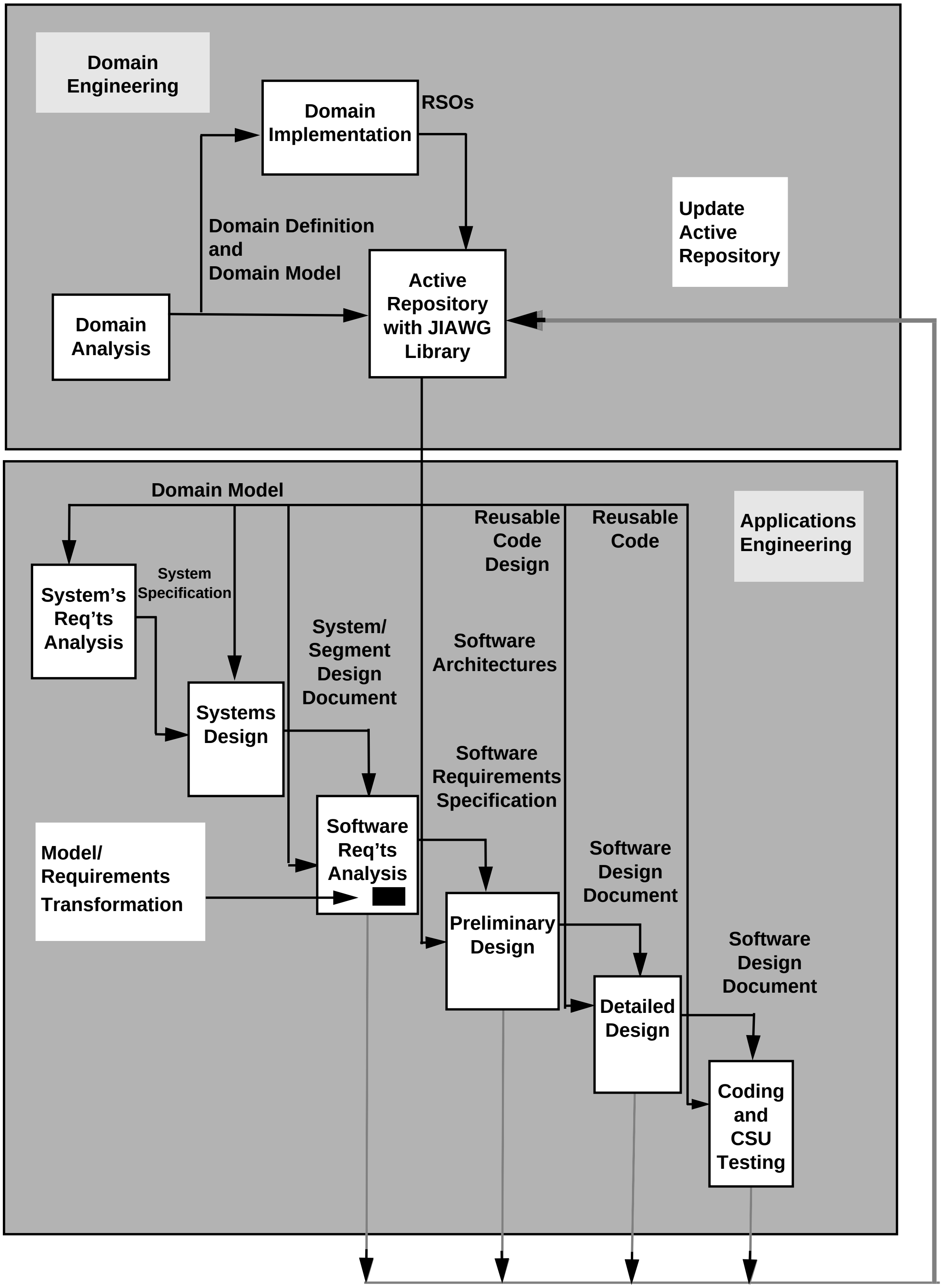


Figure 2-2 Applications Engineering Based on Domain Engineering

2.1.2 Domain Implementation

The goal of domain implementation is to produce RSOs that can be used in a DoD deliverable. The minimum items that domain implementation should produce are a reusable software architecture, reusable code designs, reusable code, and reusable tests. The reusable software architectures are a set of high level designs that can be used to implement any member of the domain family. This family includes systems which contain the minimal set of features which make sense for the domain, and elaborate systems which contain many optional and advanced features. The design information obtained and represented include tasking, data allocation, user interface, and the packaging of the requirements from the domain analysis. The roles identified in domain analysis support the definition of the user interface, and the triggers, events, and parallelism support tasking definitions. The software architectures and the rationale for selecting one over another are also recorded in the Active Repository. Traceability from the domain model to a software architecture is added to the Active Repository when the architecture is added.

Domain implementation also produces reusable code that implements the software architectures. When designing reusable code, the domain engineer selects the appropriate packaging of the tasks, data, and user interfaces identified during architecture design. The Active Repository also records traceability back through the software architecture to the basic capabilities defined in domain analysis. All the variations that are defined in domain analysis and domain implementation must be implemented to complete domain implementation. This means that there will be more code than is needed or delivered in any one system. If domain analysis identifies a set of capabilities that is not complete, then further analysis and implementation to complete the set is a possibility, since the value of the RSOs is based on their coverage of the domain. For each module, a test driver and test cases should be implemented to validate and evolve the design and code.

Code components are the primitives while the software architectures define the combination and integration of the primitives. When the variation in the architectures and designs is well understood, then the parameters and relationships can be identified and used to create tools that automate the reuse-based development process. While code components are relatively small, large-grained reusable components can be more cost effective. Large-grained components can be constructed with tools like the CAMP constructors [23]. To support large-grained reusable components, parameters and relationships must be identified. These parameters and relationships can be represented with several different technologies that support automating the engineering process. Ada does not possess the flexibility to represent all desired parameterization for reuse [MCNI86], and even where it does, automated support is warranted (e.g., CAMP Kalman Filter constructor). Before implementing RSOs or tools, the parametrization and relationships should be validated by domain experts to ensure that commonality and variation have been captured. After RSOs are implemented, they should be validated against both existing and future systems. This validation helps ensure that the RSO can be reused.

2.1.3 Active Repository

The goal of the Active Repository is to make RSOs available during Applications Engineering. The JIAWG library is a tool that will be part of the active repository [9,10]. During domain analysis, a domain definition and model are produced and stored in the Active Repository, but the model is not reused directly. The information in the model must be reorganized in accordance with DoD-STD 2167A Data Item Descriptions (DID). The tools and techniques to support the storage, retrieval, and maintenance of the domain model and other RSOs into DoD standard form are also part of the Active Repository. In Section 2.2.2, the transformation of the domain model into DoD standard form is discussed. In some cases the RSOs will be stored in libraries, but in other cases the RSOs may be supported by tools that enhance their reusability.

Several techniques can used to create reusable products for the Active Repository. The Common Ada Missile Packages (CAMP) Project has produced tools that support template completion, constraint checking (domain rules), and requirements elicitation (iteration and options) [21]. The Domain Specific Software Architectures Project at the SEI has also produced templates, tools, and techniques that support domain models [27, 19, 11]. Software Productivity Solutions (SPS) has produced a preprocessor for Ada that supports adding, deleting, and modifying capabilities through inheritance with Classic Ada (Classic Ada is a trademark of SPS) [4]. Several techniques in addition to a library can be used to make RSOs available. RSOs in the Active Repository cover the software life cycle, and traceability is defined between the RSO and the requirement(s) that the RSO implements.

The traceability is added during domain implementation because the software architectures and code do not exist during domain analysis. Furthermore, multiple implementations are possible from a single domain model, and the different implementations need to be traceable from the model. When variation exists, traceability must exist from each variation to its implementation. Also, traceability needs to exist between each gen-spec structure and its implementation. The traceability information is also used by software engineers during domain implementation in order to define interfaces and relationships. Traceability is used by the reuser to locate, retrieve, and integrate the RSOs into deliverables and to satisfy DoD-STD 2167A DIDs. Traceability defines a road map for the reuser during Applications Engineering.

2.2 Applications Engineering

For each system that is constructed using reuse techniques, the goals for that application need to be defined. These goals will depend on the coverage of the domain, the maturity of the RSOs, and the experience of the development team. For example, if there are RSOs for a short-range tactical missile system and a long-range strategic missile is being developed, then the reuse goals would be less ambitious than they would be for developing another short-range tactical missile. Our reuse goals for JIAWG include support for Pre-Planned Product Improvement (PI) and PDSS. To determine if the reuse goals have been met, data must be collected during applications engineering. Without data to measure the reuse goals, the benefits and problems can’t be identified, and the reasons for success or failure may not be determined. Another reason for collecting data is to permit comparisons of subsystems developed from reusable assets against subsystems developed without reuse. This data can be used to develop more accurate economic and planning models for reuse-based development. Once the reuse goals have been identified, data on reuse is collected for each phase of the life-cycle. Knowing the benefits and problems for reuse technology and identifying the reasons is an effective means to evolve a reuse approach. The effectiveness of reuse during Applications Engineering is difficult to measure because different applications may have different goals, and generalization of results may be impossible without a large sample.

For each reuse-based development, the Domain Engineering results must be updated. If there are changes, the domain model is updated and new RSOs may be constructed while other RSOs may be changed or deleted. The engineering of each new system and even enhancements to existing systems help identify changes to the RSOs. The Applications Engineering life cycle that is shown in Figure 2-2 is the waterfall model (although the reuse based software development concept applies equally well to the spiral model).

The domain model that is produced during domain analysis cannot be reused directly during Software Requirements Analysis because the CYOOA format is not compatible with the DoD-STD 2167A DID. If the DoD SPO can not be convinced to accept CYOOA specifications, then the domain model's specifications must be transformed into DoD Standard Requirements that can be used in a SRS. This transformation is performed during Software Requirements Analysis (SRA), and automation of the transformation is planned for OOA*Tool (OOA*Tool is a trademark of Objects, International). This transformation can be performed at any time after domain analysis, but it has been included in Applications Engineering and is discussed below.

2.2.1 DoD Standard Requirements

CYOOA assumes that the specifications produced can be used directly, but JIAWG uses DoD-STD 2167A Data Item Descriptions (DID) to define the software deliverables. CYOOA specifications must be reformatted so that they are compatible with MIL-STD 2167A Software Requirements Specification (SRS), if they are to be reused by JIAWG. Ideally, the Systems Program Office (SPO) can be convinced to make an exception to the standard and accept CYOOA specifications.

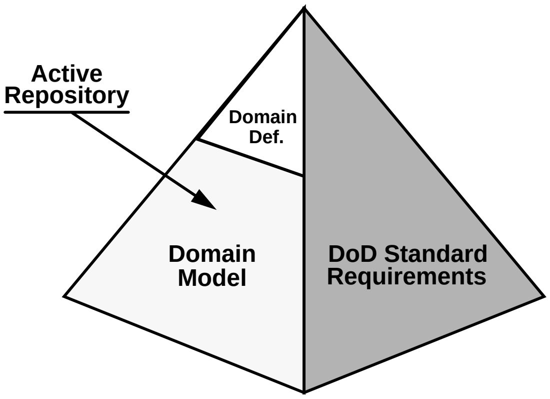


Figure 2-3 Relationship of the DoD Standard Requirements to the Domain Model

The domain model that collects, organizes, and records domain structure, classes, relationships, attributes, and services is only one representation of the domain. The more information that is captured and represented in the model, the more detail that exists for the lower level classes. The existence of greater detail for the more primitive classes gives the model the shape of a triangle. The top-level information in the model is the domain definition. Figure 2-3 represents this concept pictorially. Another representation of the domain could exist in DoD standard requirements. DoD standard requirements represents the domain as a set of requirements using DoD-STD 2167A DID format. This representation of the domain is isomorphic in most respects to the domain model, but it emphasizes the domain's functional capabilities through its interfaces. DoD standard requirements are shown in Figure 2-3 as another surface of a tetrahedron, and is another view of the domain.

Figure 2-3 does not show the specifics of the mappings from the classes, structure, relations, and services in the domain model to specific requirements paragraphs in DoD standard requirements. The details of these mappings are complex and difficult to show pictorially; a description of the mappings is given in the next section.

2.2.2 Model/Requirements Transformation

For each applicable paragraph in the SRS, the relevant information in the domain model is identified, and a means for transforming that information into DoD format is given for each applicable SRS paragraph. This discussion describes the mapping from the DoD standard requirements to the domain model, and is intended to demonstrate that the SRS paragraphs have required information in the domain model (as defined by CYOOA). Two general transformations steps that are easily recognized are deleting variations that are not needed and reformatting the information.

The first step is to make a copy of the specifications and delete all class and object variations that are not used. The variation defined in CYOOA notation is not permitted when specifications are produced for JIAWG programs. This means that the specifications must be tailored to specify only the system being developed. There is no mechanism in CYOOA to support tailoring the specifications, but tailoring is required for JIAWG use; the tailoring will have to be done manually. Some classes and objects may be deleted, and some attributes and services within objects may be deleted. The specifications that remain define the system currently being developed, and all services that remain are required to support the current domain specification or external subsystems. The rationale for the variation may specify a constraint for the use of a specification. For example, if an elevator is being specified that has no Stop Button, then that object and its specification is deleted from Figure A-11. Also, the stop service is deleted from the Elevator specification in Appendix B.3 and from Figures A-3, A-11, and A-13. The result of the first step is an object-oriented specification of the system under development. The next step reformats the specification to comply with DoD-STD 2167A DID.

In the model/requirements transformation activity, sections of the SRS for each Computer Software Configuration Item (CSCI) are created from the information in the domain model. The domain model is used to construct the following SRS sections:

- CSCI External Interface Requirements (3.1),
- CSCI Capability Requirements (3.2),
- CSCI Internal Interfaces (3.3),
- CSCI Data Elements Requirements (3.4),
- Adaptation Requirements (3.5), and
- Sizing and Timing Requirements (3.6).

The information in the paragraphs listed above can be produced directly from the model, but the model may not be useful for completing other sections. The domain model does not directly support the following SRS sections:

- Safety requirements (3.7),
- Security requirements (3.8),
- Design constraints (3.9),
- Software quality factors (3.10),
- Human performance/human engineering requirements (3.11), and
- Requirements traceability (to the System Segment Specification, Prime Item Development Specification, or Critical Item Development Specification) (3.12),

A CSCI is similar in scope to a domain, so we can relate the CSCI products to the domain analysis products. In the domain model, the information is organized by classes; in the SRS, the information is organized topically. Therefore, for each SRS topic, we identify where the information is obtained in the domain model and how it its transformed:

3.1 CSCI External Interface Requirements

The external interface requirements for Section 3.1 suggest an interface diagram that would be the domain definition’s top-level subject diagram. The interfaces in the top-level subject diagram would be labeled with a project unique identifier. CYOOA and JODA do not require or prohibit labeling the diagram. The labels must be project unique, so the interfaces labels must be checked during software requirements analysis. This section also requires that we describe each external interface. This information would come from the domain definition’s service, dependencies, and top-level whole-part diagram. References to interface requirements specifications must also be included.

3.2 CSCI Capability Requirements

The capability requirements for the CSCI (domain) are the domain definition’s services. These services are named and described in Section 3.2. The information for each capability (service) are derived from the service specification in the domain model. The following information is required for each capability (service): purpose, inputs, outputs, and mode or states. A table is created relating the capabilities by mode. The state information for the domain’s classes will contain the mode information that must be manually collected to produce this paragraph. Scenarios may be included if a requirement existed to release a bomb, since this requires several domain services to be applied correctly. This section is an expansion on the visible domain services from the domain definition.

3.3 CSCI Internal Interface Requirements

The internal interface requirements for Section 3.3 suggest an internal interface diagram that would be the structure diagrams from the domain model. An intermediate level subject diagram such as Figure A-5 in the domain model should be produced to document the major internal interfaces. This diagram should group the major subjects of the domain and identify the interfaces between them. The major services provided across subjects will be used to document these internal interfaces. The information for each interface (subject to subject) is derived from the classes' service description specifications in the model. From the listed service, the following information is extracted and provided in the SRS: a name, a service description, and the inputs and outputs of the service.

3.4 CSCI Data Element Requirements

This section is an ordered list of all data elements in the CSCI (domain). The attributes from all objects are listed in Section 3.4, but are ordered based on the three types: external interface data, internal interface data, and local data. The interface data elements require identification of the interface by project-unique identifier and references to the source and destination capability. These capabilities are the domain definition's services and are named in Section 3.2 of the SRS. This information is derived from the model's structure diagrams and the object specifications. All attributes in the domain model are included in this list.

3.5 Adaptation Requirements

Adaptation requirements are either installation dependent data or operational parameters. This information will be derived from class attributes in the model. Some attributes that maintain state data will contain installation-dependent items such as latitude and longitude. This type of information could be based on aircraft type. Other attributes may contain operational parameters such as navigation set model numbers. These attributes must be manually extracted from the model and listed in Section 3.5.

3.6 Sizing and Timing Requirements

Sizing and timing information is specified for each object and service. This information must be collected from the object specifications and combined to specify the domain (CSCI) sizing and timing requirements.

DoD standard requirements also provide leverage when they are used in applications engineering, because domain engineering creates the designs and code that implement the DoD standard requirements. The Active Repository, Figure 2-2, contains traceability between DoD standard requirements, software architectures, code, and tests. This traceability provides a road map for the software engineer when doing reuse-based development.

The model/requirements transformation is not an activity of JODA, but it produces a DoD-STD 2167A SRS directly from the domain model. This activity is shown in the Software Requirements Analysis phase in Figure 2-2, because that is when the activity is performed and the results are produced. When the user is creating DoD standard requirements, he may identify information that is missing from the domain model; this information is fed back to the Active Repository. Transformation of the domain model into DoD Standard Requirements is a task that is strongly related to domain analysis, and has been discussed to show how the results are used.

3 Domain Analysis Overview

In this chapter, we give an overview of the domain analysis process; in the following chapters, we describe the domain analysis process and products in detail. Figure 3-1 identifies three domain analysis phases, prepare domain, define domain, and model domain, which are described in this chapter. We also explain the relationship of the phases and their outputs. The majority of the analysis is done in model domain; prepare and define domain are preliminary phases where sources are collected and the problem is bounded. Domain experts are required to support each of the three phases.

Domain experts provide several important capabilities; they:

- identify source material. Domain experts are the best source for identifying software artifacts from existing systems and reference material. Frequently, the domain experts have taught classes or given presentations that describe the domain. From this material, they can identify references and provide a high-level view of the domain.
- answer questions. During the analysis process, there are issues that cannot be resolved without assistance from someone who is experienced in the domain. Questions can be submitted to the domain experts. The answers to these questions keep the analysis moving and heading in the right direction.
- identify future trends. Domain expertise includes a broad knowledge in the domain. The domain analysis team does not usually have access to information on future trends. Without this information, the results of domain analysis will not remain viable long enough to recover the cost. This information is also necessary for defining scenarios during the model creation phase.
- review results. The validation of domain analysis results is difficult. Domain experts can help to validate the results by reviewing the domain definition and model. The domain experts also can evaluate the level of detail to determine if the results will be used. If the results are too detailed and an abstract view is not produced, then the results may not be used. Having a complete package that aids the reuser in understanding the domain is all important.

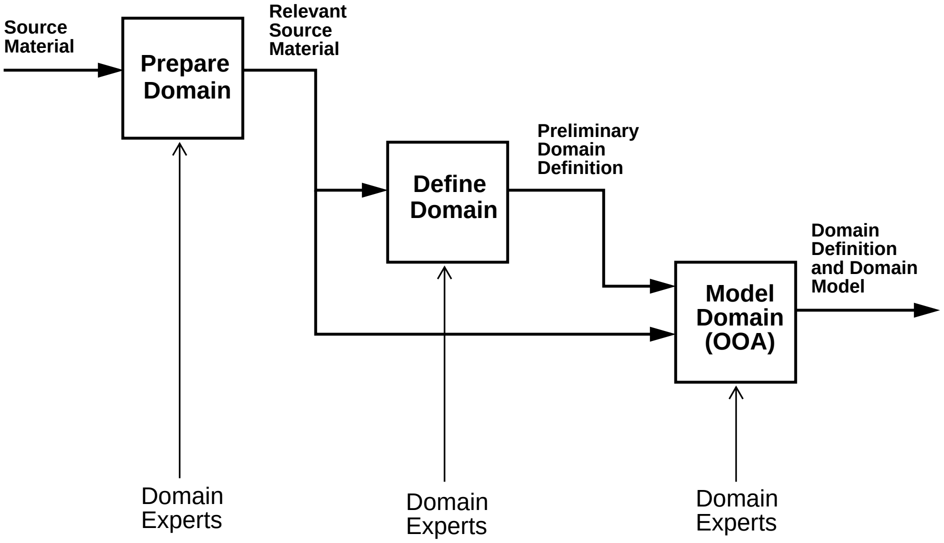


Figure 3-1 Domain Analysis Process

3.1 Prepare Domain

Domain analysis is a complex, time consuming, and detailed task that requires developing high quality abstractions of existing and proposed systems. It requires access to information from domain experts, artifacts from the development of previous systems, domain reference materials, and a knowledge of emerging technology and future trends in the domain [3]. Having access to this material requires that the domain analysis team identify, locate, and gather domain experts, artifacts, references, etcetera. The domain preparation activity that identifies and collects source material, references, and software artifacts iterates with domain definition activity. (Iteration is not shown in Figure 3-1 because it would clutter the drawing.) The source material is usually acquired over time, and the team members can produce a domain definition while they collect additional source material.

Domain preparation produces two products: domain source material and support from domain experts. Domain expertise is not the same as source material; it is help from experienced people that augments the teams knowledge and experience. Domain experts provide the services listed above; source material includes artifacts from previous systems, reference material, training material, and information on future trends that is collected and analyzed by the team. The collected source material is grist for the model creation phase. Without the source material or domain expertise, the modeling phase is impossible, and the domain analysis cannot continue, so it is stopped. The domain definition lists all source material, references, systems, and domain experts used for the analysis.

3.2 Define Domain

The domain definition is delivered and maintained in the Active Repository. (See Figure 2-2.) This top-level view of the domain uses the same notation as the domain analysis. The definition bounds the domain and is used by the analysts to clarify what is and what is not available for reuse. Potential users can quickly determine if their needs can be met by the results of the domain analysis. The domain definition includes:

- top-level subject diagram
- top-level whole-part diagrams
- top-level generalization-specialization diagram
- domain services
- domain dependencies,
- domain glossary, and
- textual description.

Finally, the domain definition identifies the context for potential reusers. For example, if we had a math library for real numbers, and a developer had hardware with only integer operations, then s/he would require an integer square root routine. The developer's searching could be limited to the domain definition, if the definition was based on data types or objects. The developer's use of the domain definition could be productive, if s/he identified and adapted the real square root routine for use. The notation chosen to represent the domain definition is the same notation chosen to represent the domain model. The domain definition identifies top-level objects and defines visible services in the domain. So, the JODA domain definition would describe all the objects as real number objects in the above example, but it would also describe the square root service. Although the techniques used to produce a domain definition are the same techniques used for domain analysis, the definition activity has been separated from analysis and included in preparation because of timing and the importance of restricting the scope of the domain. The process of domain definition is described in Chapter4.2.

3.3 Model Domain (OOA)

The JIAWG model creation phase is an extension of the CYOOA method. This phase contains three activities: examine object life-histories and state-event response, identify and walk through scenarios, and abstract and group objects. These activities are iterated since all relevant information cannot be identified in one pass. These activities are neither discrete nor sequentially occurring. The domain analysis team moves freely between activities, but the scenario and abstraction activities are more effective when a basic model has been identified. In fact, it is difficult to abstract and group objects without already having identified and defined the objects.

The first activity, examining object life-histories and state-event response, is derived from CYOOA and has not been changed. CYOOA has defined five subactivities: identify objects, define structure, identify subjects, define attributes, and define services. The analysts may perform these activities in any order, but the goal is to identify, define, and relate objects. These CYOOA activities are like scenarios but restricted to single objects; more global scenarios are used to validate and refine the domain model.

The second activity identifies, defines, and simulates domain scenarios. Experts use this basic activity to define and implement software [1]. These scenarios identify a series of services that are executed by the objects to provide high-level services to users and other systems. For example, a basic capability of a stores management system is to help the pilot release a bomb. The release process has several requirements that must be met before stores management will drop the bomb. The aircraft must be in air-to-ground mode, master arm must be selected, and a bomb and a delivery program must be selected. Once these conditions are met, the pickle button on the stick will release a bomb. Within each domain, there are many services, but to the external world there are only a few scenarios for each domain.

The final activity abstracts and groups objects so that the model is widely applicable and so that the reuser is introduced to the domain gradually. The objects in the domain can be grouped in two ways: first, by subject, and second, by whole-part structure. Whole-part structure is the decomposition technique of CYOOA, and the emphasis and discussion in CYOOA are adequate. The description of subjects in CYOOA is adequate to understand them, but the emphasis in identifying subjects is inadequate. In any domain, there are several detailed concepts the user must grasp. A high-level view of these concepts is necessary so that the reuser is not lost in the detail. Figure A-5 is such a description for OBE systems; it identifies the main elevator concepts and their relationships. Abstract descriptions of the domain are necessary to maintain the interest of the analysts, reviewers, and the reusers. Abstract views help the domain analysts to clarify their understanding. This activity depends on the existence of a basic domain model.

This model creation phase refines the domain definition and produces the domain model, that is stored in the active repository, consists of both diagrams, Figure A-11, and class specifications, Appendix B.3. This phase also adds terms to the domain glossary. Domain terminology must be defined so that the analysis team, the experts, and the eventual users will understand the information. Terminology is captured in the domain definition document and maintained in the Active Repository. In the next chapter, we describe the products that go into the Active Repository.

4 Domain Analysis Products

In this chapter, we describe the two domain analysis products: the domain model and the domain definition. The domain definition represents information that is similar to the model, but the definition serves a different purpose. The domain model is a complete view; it contains the codified knowledge that is extracted from domain experts, existing systems, and future trends, and it represents the analyst's understanding of the domain. The domain definition is a top-level view of the domain; it defines the attributes and services of visible objects, the highest level whole-part structure, a high-level gen-spec structure, and a top-level subject diagram. The domain glossary is a list of terms that are necessary to understand the other products, and it is included in the domain definition. We also briefly describe DoD standard requirements, the specification produced by the reuser during software requirements analysis. The DoD standard requirements are a separate view of the domain, but the information has been transformed so that it can be included in a Software Requirements Specification (SRS). DoD standard requirements are specifications that do not contain the variation defined in the domain model, but have been instantiated so that the variation is removed. DoD standard requirements specify a single system in the domain.

We believe that understanding the products associated with the process will provide the reader with focus. Each of the products consists of diagrams and specifications, most of which are defined in Object-Oriented Analysis [7]. Ideally, the problem space description is concise, easy to understand, and abstract, but this isn't the case for most domains. A complex detailed description is more common. Domain analysis is a labor intensive, complex process which must communicate a large amount of detailed information to a knowledgeable reuser. The domain model is the key product and will be described first, while the domain definition which is simply a high-level view of the domain will be described last. (This is not the order in which the products are produced, but the domain model is central to domain analysis. Therefore, it will be discussed first.) For the domain model and domain definition, we describe:

- the requirements that must be satisfied, and
- the notation used to define the product.

4.1 Domain Model

The domain model is used to collect, organize, and represent all domain information. The information is collected using diagrams and specifications defined by CYOOA. Several types of diagrams are included in the domain model, such as gen-spec diagrams, whole-part diagrams, state-event diagrams, scenario diagrams, etcetera. After the initial analysis, the domain model can be modified to update the domain analysis results. Its primary use, however, is to provide a framework in which to collect and organize information. Before we describe the domain model in detail, we list the requirements that it must support.

4.1.1 Requirements for the Domain Model

The purpose of the domain model is to represent the problem space. It defines all the domain capabilities and their variations and combinations. To help the user understand the domain, the model presents an abstract view that identifies the significant relationships and services. To help the user apply the capabilities, the domain model must describe how the capabilities are related through scenarios, and the rationale for how and when to use each capability. Since we want the capabilities to be widely applicable, they should also be abstract. Therefore, the domain model must describe:

1. the domain's major parts,
2. the objects and abstractions (classes),
3. the relationships between these objects or abstractions (classes),
4. the attributes and constraints on the abstractions (classes),
5. the services that are provided by these abstractions (classes),
6. scenarios that define the more dynamic aspects of the domain, and
7. the rationale for choosing an instance, option, or variation over another (rules of thumb). Rationale includes the risks, trade-offs quality issues (performance, portability understandability, etcetera), and scenarios of use.

The domain model requirements are listed above, but some quality factors are important and need to be acknowledged. These qualities are necessary if the domain model is to be effective and remain viable over changes in technology, time, needs, people, and budget. To support these changes the domain model must be:

- adaptable
- understandable
- usable,
- correct, and
- maintainable.

To meet the requirements listed above for the domain model, we must analyze, organize, and represent a large amount of information. The organization of this information is done graphically, but the object specifications are textual. Below, a description is given for each domain model diagram, and we identify the requirements that the diagram supports. These descriptions are listed in order of importance. The class is the basic element of CYOOA and the domain model; the structure and subject diagrams define the domain structure; and, the scenario diagrams describe the dynamic aspects of the domain. The domain model includes:

- class specification. Class diagrams define the class, their attributes and services [7]. Class diagrams make a contribution to solving domain model requirements 4, 5, and 7.

- structure diagrams. Structure diagrams identify the commonalities, abstractions, and variations of the domain classes, and identify complexity in the domain [7]. Structure diagrams contribute to meeting domain model requirements 1, 2, and 3.
- subject diagrams. Subject diagrams identify the amount of information that is presented to the user at one time. The grouping of objects is a kind of abstraction, and subject diagrams also identify interfaces between these groupings. They address requirements 1 and 2 for the domain model.
- scenario diagrams. Scenario diagrams identify the high-level services that the domain provides, and they bind the domain services into comprehensive groups. Scenario diagrams define the domain event response and life history information similar to object life histories of Coad and Yourdon [7]. Scenario diagrams address requirement 6 for the domain model.

4.1.2 Class Specifications

Class specifications are a major element of the domain model. They don't define the domain structure, but they do define the details. Class specifications are more textual than graphic. They are template driven and a formal language, as recommended by Coad and Yourdon [7]. These class specifications list and define their attributes, connections, constraints, rationale, and services. As the user works within the domain, the services become more familiar, and the need to refer to them decreases. Also, the names of the attributes and services may be sufficiently mnemonic so that they can be understood without detailed specifications. In Appendix B.11, for example, the service ring for the Elevator Arrival Bell does not need to be defined. In Appendix B, examples of class specification are given; full descriptions are given in Chapter 3 of Object-Oriented Analysis [7]. Class specification parts are described below in order of importance. Attributes and services define the class; the state diagrams provide more detailed information on that definition.

4.1.2.1 Class Attributes

The attributes of a class define the data which is used to describe an instance of a class. Frequently, this data needs to be retained over time. In this sense, class attributes are very similar to class variables from Smalltalk. (Smalltalk is a trademark of Xerox.) In the specification, we identify the attribute and its type, define its range constraints, identify any rationale, and a textual description. For attributes that are used in an external interface, we must identify the visible service (source capability) and its name (project unique identified), and name the attribute with a project unique identifier. For the attributes that are used in an internal interface, we must identify the domain service and its name (project unique identifier). We must also identify the visible service that it supports. For example, #_elevators in Appendix B.2 is an attribute that defines the state of the class of type integer of range 1 to max_#_of_elevators; it is the number of elevators that is accessible from the lobby.

4.1.2.2 Class Services

The services of a class define the processing performed on or by the class when requested or triggered. These services can be requested by other classes or caused by events or users of the service. For example, in Appendix B.2, the request_elevator_up service comes from the Request Elevator Up Button. If the button is not already on, the request is sent to the Controller. The lack of a service description implies that this is a simple service for which a detailed description is not required.

4.1.2.3 State Diagrams

State diagrams are useful for defining system behavior, event-response, concurrency, and the order of processing. These diagrams are not required by Coad and Yourdon, but they may be used for more complex systems to describe system behavior (system dynamics). State diagrams, as defined in Statemate [15], are effective for specifying real-time performance requirements.

4.1.3 Structure Diagrams

Gen-spec and whole-part diagrams describe the complexity in the problem space. These diagrams define what is common and what is different in terms of classes and structure. Whole-part diagrams define the composition of the domain while the gen-spec diagrams identify the variation in classes; these diagrams define the structure of the domain.

Another feature of the gen-spec diagram is to indicate that a relationship for specific instances does not exist. For example, in Figure A-10, the Top Floor Elevator Lobby does not have a relationship with the Elevator Up Arrival Light. This light means that an elevator is at this floor going up. One can not go up from the top floor, so this connection is not present. This is indicated by an “X” at the beginning and end of the relation line. The implication of the information in Figure A-10 is that the Top Floor Elevator Lobby does not have an Elevator Up Arrival Light and Up Request Button and does not provide a req_elev_up service, and the Bottom Floor Elevator Lobby does not have a Down Elevator Arrival Light, and a Down Request Button and does not provide a req_elev_down service.

4.1.4 Subject Diagrams

Subject diagrams provide a means for controlling how much of the domain model the reader considers at one time. The highest level subject diagram is included in the domain definition. Other lower level views are also possible (see Figure A-5). In a large domain, subject diagrams that group the major elements of the domain are a necessary and an effective mechanism for controlling how much of the domain the user must comprehend at one time. If the reuser or the domain experts cannot examine the domain a piece at a time, they may conclude that it is too complex and not worth their time. Using subject diagrams to present a simplified view of the domain will help reusers and reviewers to understand the domain model. In the OBE systems example, two levels of subject diagrams have been defined. At the highest level, Office Building Elevator Systems provide services to Users. The more detailed subject diagram (see Figure A-5) has seven objects (Users, Service Button, Status Indicators, Elevators, Elevator Lobbies, Doors, and Controllers). The Users interface only with the Service Buttons and Status Indicators, while the Controller and the Doors interface with Elevators and Elevators Lobbies. The Elevator Lobbies and the Elevators also interface with the Service Buttons and Status Indicators. A simplified view is essential for users to understand the domain. The amount of detail in even small domain is so large that one must have a simplified view to get started. This simplified view of the domain should contain seven plus or minus two subjects; this simplicity makes it easier to understand the details.

4.1.5 Scenario Diagrams

Scenario diagrams capture the dynamic aspects of the system. Two possible notations for scenario diagrams are R-nets as used in SREM [2] and statecharts [15]. The notation proposed by Coad and Yourdon does not have a mechanism for defining performance requirements across classes or for showing concurrency if it is required. Statecharts have the ability to specify these requirements and are used to define scenarios in JODA [15]. Domain services are used by applications engineers to support larger requirements; stores management supports bomb targeting and release. These combinations of services need to be documented for the applications engineer who uses the domain analysis results. StateCharts are being used to define these systems. Scenarios also help reduce the amount of complexity the reuser or domain expert must comprehend, and they also define the context for domain objects and services.

4.1.6 Evolution of the Domain Model

During the development of new systems from RSOs, there are requirements that are not covered by the domain model. These requirements must be included in the domain model, and the appropriate reusable artifacts modified. Changes that result from applications engineering fall into the same classes that are described above. Class attributes or services may be added or modified, or changes may be made to the whole-part structure. We may also add new objects to the domain model causing the biggest change to the domain model. For example, if the system being engineered has new requirements, these must be included in the domain model along with the appropriate update to the domain definition. The design and implementation phases of domain engineering must also be revisited to modify the RSOs that support new requirements. This evolution takes the domain model full circle. The domain model is created during domain engineering, used during applications engineering, and updated again on the basis of the changes that are identified during applications engineering.

4.2 Domain Definition

The purpose of the domain definition is to help the user determine if these domain analysis results will be useful in the new system that he is developing. The domain definition is a top-level view of the domain, and it describes the context in which the domain is useful. The domain definition answers two questions:

- Does this domain provide capabilities that I need in the new system?
- Can this domain be adapted to work in the context of the new system?

Consider again, the need for an integer square root function when we only have real number math routines. The domain definition for a math subroutine library (absolute value, floor, ceiling, square root, trigonometric functions, and logarithmic functions) should describe the following for the user: the data type, the function, the constraints on using the routine, and dependencies such as the computational method and the precision of the machine. The description of the data type would tell us that we don't have any integer functions, but we do have a real square root function that might be adapted. The data type, function, and constraints answer the first question; the dependencies, data type, and constraints answer the second question.

4.2.1 Requirements for the Domain Definition

The domain definition is used to determine whether or not the new system is within or related to the domain. The domain definition identifies the relationship of new problems to the results of domain analysis. To answer the two questions identified above, the domain definition defines:

1. the domain interfaces,
2. the services the domain provides through its interfaces,
3. the domain boundaries (i.e., what domains are very similar, but what attributes and services determine inclusion and exclusion),
4. the source material, references, and domain experts consulted to produce the domain model,
5. the major elements or subdomains of the domain,
6. the terminology used to define the domain, and
7. the domain dependencies (i.e., the services provided to the domain through its interfaces)

The terms that are defined during domain analysis are necessary for the user to understand the domain concepts, the domain definition, and the domain model. Many organizations define and use local terms (that have evolved over many years) for concepts, abstractions, services, and constraints. These definitions are not equivalent from one organization to another. The domain terminology should define the relevant terms, concepts, constraints, equations, and constants.

The domain definition requirements listed above have been identified from the planned use of the domain definition. From these requirements, we have identified a set of diagrams that meet those needs. Context description is another term which is frequently used to describe the setting in which a domain will function. The use of context description has been derived from the top or zero level data flow diagram of structured analysis. This diagram identifies the external interfaces which define the system context. Domain definition is preferred over context description and will be used in this document because the user needs to know more than just the context. The user needs to know what is defined as in the domain. The domain definition includes:

- Top-Level Subject Diagram. The Top-level subject diagram identifies the other systems or subsystems with which the domain interfaces. This diagram helps meet requirement 1 above.
- Top-Level Whole-Part Diagrams. These diagrams define the whole of which the domain is a part, and the parts when the domain is the whole. These diagrams help to meet requirements 1, 2 and 5.
- Top-Level Gen-Spec Diagram. This diagram identifies the attributes and services that distinguish this domain from similar domains. This diagram helps to meet requirements 2 and 3.
- Domain Services. A list of the services which are visible to external systems and subsystems. These service definitions satisfy requirement 2.
- Domain Dependencies. A list of the services that must be available from other systems or subsystems for the domain to meet its requirements. These dependencies satisfy requirement 7.
- Domain Glossary. The terms that are used to define domain concepts are essential for the reuser to understand the domain. Terminology evolves over time and represents an organizational view of the domain. These terms must be documented to completely and accurately describe the domain. The glossary satisfies requirement 6.
- Textual Description. The text binds these lists and diagrams together into a integrated view. The textual description, that includes: a list of the source and reference materials used by the analysts; a list of software life-cycle objects from systems examined; and a list of domain experts and their areas of expertise, should be restricted to a few pages. This description satisfies requirement 4, but it also supports all the other requirements listed above.

4.2.2 Top-Level Subject Diagram

The subject diagram identifies the domain interfaces. If the domain is itself a whole system, then this diagram is its external interfaces. On the other hand, if the domain is embedded in a larger system, then these are its interfaces to other subsystems and the external world. An example is given in Appendix B.2. The domain in this example is Office-Building Elevator (OBE) Systems. These OBE systems interface with users and moveable equipment. Users are transported by and receive status from the elevator system, while the equipment is moved by the elevator system. In data flow diagrams (DFD), the highest level (zero level) diagram has a specific name; it is called a “context diagram.” The top-level subject diagram provides the same information as the DFD’s context diagram because it identifies the domain’s external interfaces; it doesn’t identify how other domains are related to the system. Related domains are identified from the top-level whole-part diagrams.

4.2.3 Top-Level Whole-Part Diagrams

Normally, there are two whole-part diagrams. The first defines the whole in which the domain is a part; the second defines the parts of the domain.

Figure A-2 defines the relationship of the OBE systems domain to office building transportation systems domain. An office building has elevators, stairs, and possibly escalators. There may also be ramps, or moving sidewalks depending on the precise definition of the higher level domain. Today, single story office buildings may not be very common, but they are possible. Therefore, the parts (elevators, escalators, and stairs) in Figure A-2 are optional. Also, we may have more than one elevator system, escalator system, or set of stairs within an office building. Therefore, multiple instances of the parts may make up the whole.

In the context of an office building, it doesn't make sense to have an escalator, stairs, or elevators without an office. It also doesn't make sense to have an elevator, escalator, or stairs serve more than one building because these elevators, escalators, and stairs have to be physically located in the building. There is one and only one Office Building Transportation System for each elevator, escalator, or stairs.

Figure A-3 defines the parts of the OBE systems. OBE systems are composed of a set of elevators, a controller, and elevator lobbies. The connections in Figure A-3 define the range of the whole-part relationships. An elevator system is composed of at least one elevator and two elevator lobbies. A controller is optional because a system with one elevator and two lobbies doesn't require a controller to provide fair and equitable service.

An elevator, controller, and an elevator lobby are the parts of the elevator system. An elevator or a controller may not be part of more than one elevator system, but an elevator lobby may be part of more than one system when the elevators are grouped to serve a subset of the floors in a skyscraper. These relationships are defined in Figure A-3.

After the whole-part diagrams have been examined, we can see that defining the context for our domain is helpful. We are not looking at elevators in malls, shopping center, factories, airports, or warehouses. We are examining elevators in the context where people are the primary users. Elevators in factories and warehouses move equipment or inventory. Elevators in malls, shopping centers and airports primarily move people, but the traffic flow is regular, and the people are frequently carrying something. By examining the whole-part structure of the elevator, we can see that these OBE systems have not included escalators in the domain. Some office buildings may have shopping facilities in the lower floor and use escalators for transportation between those floors. These escalators are not part of the OBE domain. Therefore, the elevator system for a building with shops on the first two or three floors that are serviced by escalators is related to but not included in the OBE domain because it has escalators, which are not in the domain. We might consider the combined elevator escalator systems as a building transportation system, but it would be a different domain.

4.2.4 Top-Level Generalization-Specialization Diagram

Within the domain definition, the classification (in CYOOA, generalization- specialization diagrams are also called classification diagrams) diagram identifies the commonality and variability at the top level of the domain. Classification structure also helps to define the domain boundaries. The classification structure does not accomplish this alone, but in cooperation with the other diagrams in the domain definition. Figure A-4 defines a classification structure on Elevator Systems. The common attributes and services are given in root class Elevator System.

In this classification structure, four attributes of building elevator systems have been identified. They are: #_elevators, #_elevator_lobbies, capacity, and destination_set. Two services have also been identified: travel_up and travel_down. Each elevator system in the class shares the attributes and services defined in the root of the gen-spec structure unless the attributes or services are excluded by an “x.” In this case, no attributes and services are excluded, but specific values are given which define the instances of the class. In Figure A-4, for example, in the Penthouse Elevator System, there are only two lobbies and one elevator. Also, the services are restricted from travel_up and travel_down to go_penthouse and go_lobby. The Freight Elevator System has two additional attributes: padding and manual control. These attributes distinguish the freight systems from the more general Building Elevator System. Another difference is the hold service for the Freight Elevator System. Further extension of this gen-spec structure is possible, but it does not serve the purpose of this document to fully detail that structure. This domain analysis is concerned with OBE systems; each of the classes in Figure A-4 that is not included in the domain is marked with an “x” preceding the class name. This notation further clarifies the domain definition.

The common services provided by the domain’s classes are important for defining the boundary. If the utility of the domain is being evaluated, then the user needs to determine whether or not the services from the domain’s classes meet the requirements of the new application. If they do not, then updating the domain model is one possibility. Another possibility is using only those portions that apply. A very important benefit of the gen-spec and whole-part structure is the ability to identify the applicability of partial solutions, since some new systems will lie outside the original domain boundary. The full specification of the visible domain services is included the domain model.

4.2.5 Domain Services

In the domain definition, the domain services are used to determine whether or not a new system’s requirements can be adequately supported by the results of a previous domain analysis. This means that the services provided at the domain interface must meet the requirements of the new problem statement. In the OBE systems in Figure A-1, the highest level interface is between the user and the Elevator Lobby and Elevator classes. This dual interface implies that some subset of the services provided by those classes defines the domain services. From an examination of Figure A-12, the services available from the Elevator Lobby are: req_elev_up, req_elev_down, open_doors, and close_doors. Also, from an examination of Figure A-11, the potential services from the Elevator are: status, sound_alarm, close_doors, open_doors, select_dest, signal_arrival, unlock_reardoor, lock_reardoor, and stop. After examining these lists based on the subject diagram in Figure A-1, the elevator user has access to the following services:

- req_elev_up from Elevator Lobby,
- req_elev_down from Elevator Lobby,
- sound_alarm from the Elevator,
- open_doors from the Elevator,
- close_doors from the Elevator,
- unlock_reardoor from the Elevator,
- lock_reardoor from the Elevator,
- stop from the Elevator, and
- select_dest from the Elevator.

These services are the same services defined by the Service Button class defined in Figure A-9. One way to present the services in the domain definition, is to define a subject diagram that includes all the classes which provide a domain service. Another effective means for identifying visible services is from the scenarios defined for the model. In the OBE example, we have an abstraction, Service Button, which achieves the same effect.

4.2.6 Domain Dependencies

It is not sufficient to document only the boundary of the domain. Basically, all domains depend on:

- the availability of services from external objects,
- the correct performance of services in the domain's parts,
- the maintenance of the system,
- the availability of runtime resource, and
- the development environment support that is needed to use the domain analysis results to construct a new system (does the compiler support the necessary options?).

The dependency description helps to define the context for evaluation and application of domain RSOs. If we consider applying the RSOs derived from a domain analysis to the development of another system, we must be able to determine whether or not the RSO will function correctly in the system under development. Meeting the domain dependencies is an essential part of being able to reuse RSOs; domain dependencies are the requirements which must be met for the RSOs to function correctly.

For the OBE systems domain, the dependencies are:

- availability of electrical power,

- physical and structural properties of the materials used for construction,
- structural integrity of the building,
- maintenance of the system.

In this section, for example, the dependencies of the OBE systems are matched with the general classes listed above. Not all dependencies apply. There is no external interface, so there is no dependency on external services. The subdomains of the OBE systems domain do not add any dependencies. The OBE systems are also dependent on proper maintenance of the hardware, motor, cables, and wiring. The availability of power and the integrity of the building are both part of the runtime environment. The physical and structural properties of the materials used to construct the system are examples of the development environment.

4.2.7 Domain Glossary

The terminology produced from the review of the source material and references is collected and provided in the domain definition document. There is no CYOOA notation to capture the terminology, but the terminology should be included as a glossary in any report that documents the model. The domain glossary should be available so that a potential user can understand the domain definition and the domain model. Concepts which are essential to the domain need to be identified and recorded; any terms which need clarification should also be included. In Figure A-12, for example, the Elevator Up Arrival Light in the Elevator Lobby indicates that the elevator is located at this floor, and the light is on from arrival through departure. The Elevator Floor Location Light in the Elevator in Figure A-11 is on from the time it arrives at that floor until it arrives at another floor. Thus, the semantics of the two are different for being on a floor; these differences need to be identified.

4.2.8 Textual Description

The purpose of the textual description is to bind the diagrams and lists together into one document; the goal of this definition is to help potential users determine whether or not their requirements can be satisfied by the RSOs of the domain. The description also lists:

- the source material (existing systems documentation) that was used for the analysis,
- the sources for, and descriptions of, future trends in the domain,
- the domain experts, reviewers, and reviews of the domain material, and
- the domain mission, i.e., the operational context that it supports.

The textual description provides an integrated view of the domain and helps the reader to make the transition from one diagram to the next. Another benefit of the domain definition document is that it can easily be disseminated, reviewed, and updated. It is essential that the domain definition be kept current through continual review and update.

Adaptation					
Types	**Common Classes**	**Classes by Restriction**	**Classes by Additions and Deletions**	**Structures by Iteration**	**Structures by Additions and Deletions**
Kinds of Adaptation	No changes to RSOs in any form.	Attributes and Services are restricted to specific instances.	Attributes or Services are added to or deleted from the class.	The number of structural subunits of an object can vary.	The presence of a structural subunit is an option.
Examples	Elevator Arrival Bell is common to many elevator systems.	Destination Selection Button is an instance of a Service Button.	Elevator Rear Door has added services lock and unlock.	Elevator has one or more Elevator Destination Lights.	Elevator has an optional Stop Button.

Figure 4-1 Table of Adaptation Types

4.3 DoD Standard Requirements

DoD standard requirements are the artifacts derived from the domain model for the development of new systems using the results of domain analysis and domain engineering. DoD standard requirements are a DoD formatted set of domain requirements-specifications that have been tailored to specify the requirements for a single system covered by the domain definition. The domain model includes abstract, variable, and optional specifications while the DoD standard requirements contain concrete specifications. The domain model supports ideas that are similar to templates, macros, and generics, where the decision and a range of selections are known ahead of time. In the case of DoD standard requirements, the parameters and possible selections are defined in the domain model and the selections are made so that a member of the domain is specified. This approach to reusable requirements is a combination of requirements- specifications and requirements adaptation. Figure 4-1 gives an overview of the types of adaptation possible in the domain model to support variation in both classes and structure.

5 Prepare Domain

In this and the following two chapters, we describe the domain analysis phases in detail. For each phase, the purpose of the phase, the inputs to the phase, the outputs from the phase, and the review at the end of the phase are defined. In some cases, the phase's input or output may be more than data. For example, the output from a step can be to obtain support from a domain expert to review and comment on the domain analysis results. Figure 3-1 provides an overview of the entire domain analysis process while Figure 5-1 identifies the two domain analysis preparation activities: acquiring domain expertise for the analysis and collecting source and reference material.

The results of the preparation phase are reviewed at the end of the definition phase to ensure their accuracy, reality, and viability for reuse. The domain definition phase is iterated with the preparation phase because obtaining information, interacting with domain experts, and reviewing the definition are essential to providing a precise definition. Since it takes time to identify and acquire the required sources, the analysts will not be fully occupied during the identification of domain expertise and source information and can begin the analysis by defining the domain. The domain definition controls the scope of the analysis effort, and may be changed during the domain modeling phase, but an early definition is necessary to focus the team's activity. Before defining the domain, sources for domain analysis are identified, located, and collected.

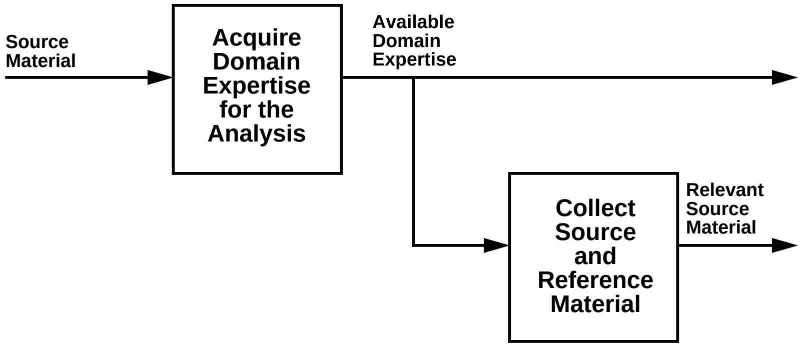

Source Material
Acquire Domain Expertise for the Analysis
Available Domain Expertise
Collect Source and Reference Material
Relevant Source Material

Figure 5-1 Domain Preparation Process

Figure 5-1 identifies the activities for gathering domain source material. The purpose of these activities is to identify, locate, and obtain the domain expertise, reference material, and system artifacts for the domain modeling. This activity is executed in parallel with domain definition, and the review at the end of the domain definition activity examines the material collected and expert support obtained in order to determine if the support is adequate.

The inputs to domain preparation are the available source material. The organization's business plan helps to identify domain experts who recommend domain reference material and documentation on existing systems. These experts are the best source for identifying future trends in the domain because they have knowledge of, and experience with, several systems. Domain experts also identify future trends that are necessary to keep the results from becoming obsolete.

The outputs from the preparation activity are the sources and the material for available domain expertise and relevant source material.

JODA recommends reviewing domain documentation from three existing systems because the domain analysis should cover as much of the domain as possible. Analyzing three previous systems will not guarantee that coverage, but only examining one system may not provide enough coverage. Examining two systems is better than one, but three provides even more breadth. Examining more than three existing systems could be productive, but the analysis may not be cost effective. We rely on the experts to identify areas that are overlooked. We use analysis (reverse engineering) to identify detailed information and confirm expert views.

5.1 Acquire Domain Expertise for the Analysis

JODA requires the availability of domain expertise. Three potential sources for identifying domain experts are:

- upper-middle managers,
- chief architects from previous developments, and
- chief architects from current developments.

To obtain this expertise, senior management must support the need to make a domain expert's time available. When the experts are identified, they are informed of the notation that will be used so that they may become familiar with the CYOOA notation. Support from domain experts must also be obtained because the results will be detailed and require a concentrated effort from the domain expert to perform the required review. Also, the domain analysis team must concentrate on producing results that can be quickly and easily understood. If domain expertise cannot be obtained through this or alternative means, the domain analysis should not continue because the results can not be adequately validated.

5.2 Collect Source and Reference Material

Domain experts can identify and recommend applicable sets of system documentation and domain reference material for the analysis team. After a list of pertinent reference material and domain documentation has been compiled, the materials are evaluated. The quality and coverage of the system artifacts are the factors that should be used to select information for domain analysis. An optimal set of documentation would include broad, high quality coverage. In reality, both quality and coverage may be lacking. Obtaining these materials will be aided by the support of senior level management. With senior level management support and guidance from the domain experts, the analysis team gathers the necessary reference material and system artifacts.

Obtaining access to classified or proprietary material can be a problem, so unclassified and nonproprietary sources and references are preferred. The classification of a domain is a real hindrance to open discussion and review. Regardless of the classification and sensitivity of the material, the appropriate source material and reference must be obtained for the domain analysis to be successful.

Another potential source of information is from previous domain analyses. These efforts may not have been called “Domain Analysis”, but they may contain essentially the same “Generic” or “High Level” information. Domain experts should be queried as to whether or not they are aware of previous studies that have attempted to capture, analyze, or represent information about the domain. These previous studies can provide an excellent starting point for further analysis.

The domain preparation activity occurs in parallel with domain definition, and the decision whether or not to proceed with domain modeling is made at the review. If the resources for the domain analysis are determined to be insufficient by the domain experts, then the proposed analysis must be re-examined with the sponsoring agent.

6 Define Domain

In this chapter we describe the domain definition activity. Figure 6-1 gives an overview of the domain definition phase. The definition is used by potential users of the domain analysis results to determine if their requirements can be met by the results of a particular domain analysis.

The source and reference materials obtained during the preparation activity are analyzed to create a domain definition that is refined during domain modeling. The better the definition in the early phases, the easier the successive phases will be. From the domain selection process and from discussions with the domain experts, the team will have started to form a domain definition, but it is necessary for that definition to be documented, reviewed, analyzed, and updated.

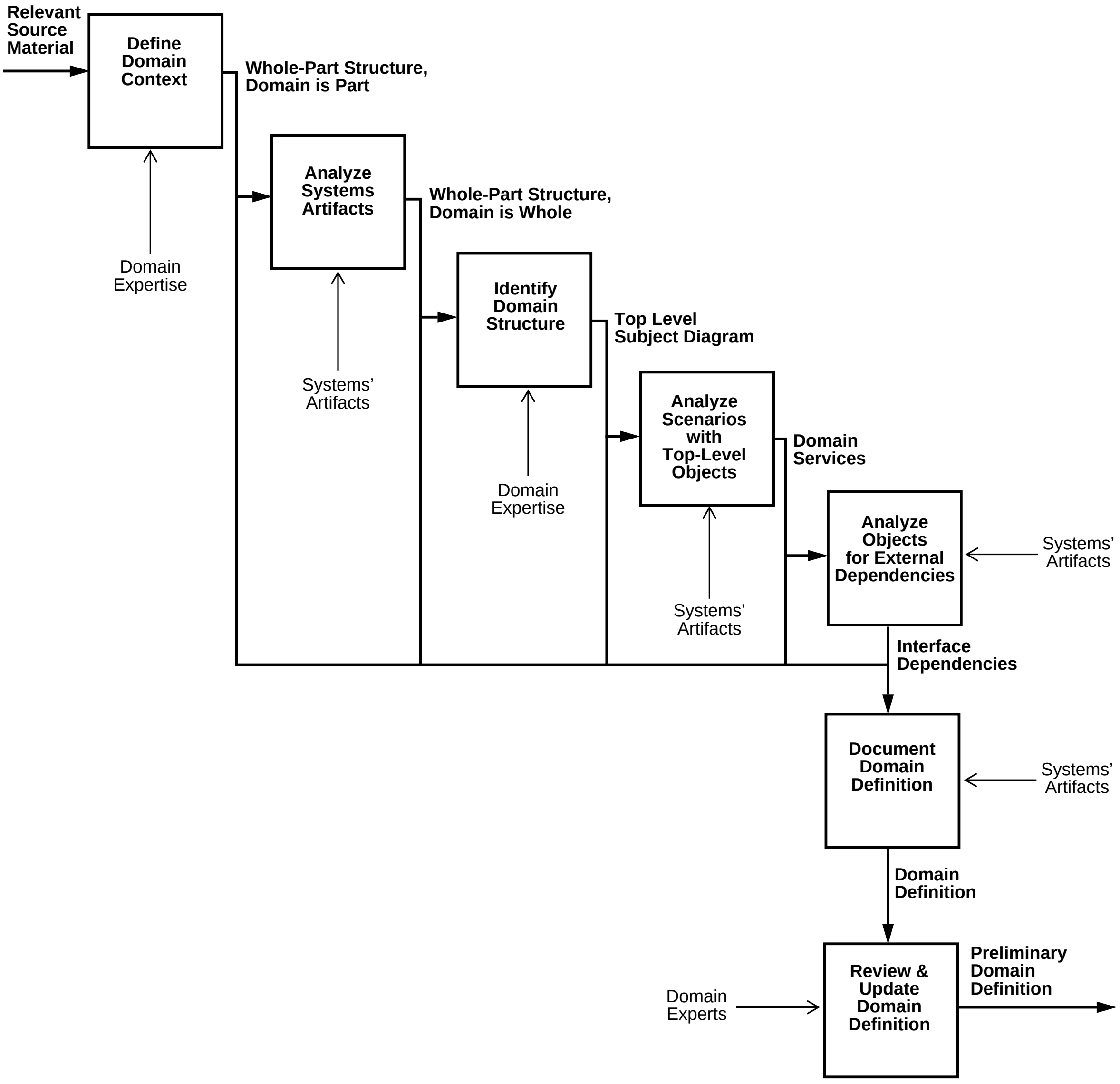
The output of this activity is a domain definition which is described in detail in Section 4.2 above. The preliminary domain definition will not contain all the detail and precision that will be present in the final version, but the following outputs are produced:

- top-level subject diagrams,
- set of whole-part diagrams,
- domain services,
- domain glossary,
- domain dependencies, and
- textual description.

6.1 Define Domain Context

Initially, the domain context needs to be defined. It is easier to retrieve that information from a domain expert than to obtain it from system documentation. The domain context may not be easy to identify from existing systems because the domain may not map directly onto the structure of previous systems. While interviewing an expert, the domain analyst should ask the expert to describe the major subsystems or systems with which the domain will interface. Do these other systems/subsystems provide service to the domain being analyzed, does the domain being analyzed provide them services, or are both true? The experts should also be asked to characterize that interface: is this a common interface? Is the interface necessary or optional? What domain requirements can be linked to this interface and service? If the domain directly supports a user, then it is useful to review operators' manuals, operations manuals, and service manuals.

To document the context for the domain, JODA requires a high-level subject diagram which identifies the customers for the domain services and shows the domain context, i.e., what other systems use its services. This diagram only identifies that there is an interface; it does not identify the customers for each domain service. The domain customers may be users, other programs, or applications. To validate the subject diagram, the domain analysis team will examine artifacts from current or previous developments. If they extract the information from documentation, then they should review the information with a domain expert. It is important to identify other systems and subsystems which obtain services and provide services to the domain. A first cut at this diagram can be made quickly, because as the analysis continues, it will be refined based on analysis of existing systems.



Relevant Source Material
Define Domain Context
Whole-Part Structure, Domain is Part
Domain Expertise
Analyze Systems Artifacts
Whole-Part Structure, Domain is Whole
Systems' Artifacts
Identify Domain Structure
Top Level Subject Diagram
Domain Expertise
Analyze Scenarios with Top-Level Objects
Domain Services
Systems' Artifacts
Analyze Objects for External Dependencies
Systems' Artifacts
Interface Dependencies
Document Domain Definition
Systems' Artifacts
Domain Definition
Domain Experts
Review & Update Domain Definition
Preliminary Domain Definition

Figure 6-1 Domain Definition Process

6.2 Analyze Systems Artifacts

JODA also includes in the definition a whole-part structure where the domain is one part. This whole-part diagram is a variant of the same information presented in the subject diagram above, but when completed, the whole-part diagram identifies services obtained from outside the domain. The context for the domain being analyzed need not be fully described, but the whole-part part diagram does differentiate between peer level subsystems/systems and larger systems or aggregates. Again, this is not a detailed figure; it need not show all services or attributes for external objects.

6.3 Identify Domain Structure

JODA also includes in the definition a whole-part diagram where the domain is the whole. To derive this whole-part structure, a domain expert is interviewed. The information could be obtained from system artifacts, but again, it is easier to obtain the domain structure from an expert. In fact, the same expert does not need to provide both whole-part structures. One of the constant difficulties the team will face is resolving different views of the domain. Therefore, resolving two different initial views will provide practice.

The analysis team will require access to at least one set of system artifacts and one domain expert to produce the subject diagram and the whole-part diagrams. Access to more than one set of documentation is recommended, but since this activity is occurring in parallel with identification of domain source material, one set of documentation may be all that is available. The most appropriate software artifacts to review are systems and software requirements specifications, high-level software designs, and operations manuals.

6.4 Analyze Scenarios with Top-Level Objects

To identify domain services, scenarios are defined that utilize the high-level domain services. To define scenarios, systems artifacts are analyzed and domain experts are interviewed. Scenarios are very useful for understanding systems [1], and they can be extracted directly from existing documentation, like the Tactical Operations Manual for the F/A-18 that describes in detail the procedures the pilot uses to manage his weapons. These procedures correspond directly to services of the domain. In this early activity, one must consider the time spent analyzing scenarios, and it must be balanced with preparation effort which is occurring at the same time. Again, it is easier to obtain scenarios from domain experts than to define scenarios from systems artifacts. Identifying services may also be accomplished by examining an object's life history, which is a different kind of scenario.

6.5 Analyze Objects for External Dependencies

The final step in creating the domain definition is to identify domain dependencies by examining each of the requirements listed in Section 4.2.6, Domain Dependencies, and to determine how each requirement is applied to the domain. After recording the domain dependencies, the analysis team is ready to document the domain definition for review by the domain experts.

6.6 Document Domain Definition

The diagrams and services which have been derived by the domain analysis team do not convey enough information. The value of such information is enhanced when it is put in context with a textual description. The domain definition document is a living document which is reviewed throughout the analysis and is delivered with the RSOs and domain model at the conclusion of domain engineering. Specifically, the document will list and define each domain service, and each external dependency. The recommended order for these diagrams is:

- introduction containing references, reviewers, and background,
- whole-part diagram, where domain is a part,
- top-level domain classification diagram (not included in initial draft),
- top-level subject diagram,
- domain services,
- whole-part diagram, where domain is a whole,
- domain dependencies, and
- domain glossary as an appendix.

JODA also includes a gen-spec diagram in the domain definition, but it is too early in the domain analysis to accomplish this with any confidence. The gen-spec diagram requires access to multiple system definitions or extensive discussions with a domain expert to draw out this complex information. A top-level classification structure will always be produced during the creation of the domain model.

6.7 Review and Update Domain Definition

The domain definition process does not terminate until a successful review is held at the end of the phase. Figure 6-2 identifies the activities of the review process whose purpose is to refine the domain definition based on the knowledge of the domain experts, and to determine if the domain analysis team has the necessary resources to perform the analysis. Reviewing the domain definition is especially important since the definition controls the scope of domain modeling that is about 75 percent of the analysis effort. The domain analysis team reviews the definition with the experts, and the analysts describe their analysis approach. This approach must be accepted before beginning model creation.

The domain definition document, references material, and documentation of existing systems are the inputs to the review and update activity. The output from this review is a preliminary domain definition, used for establishing and maintaining the focus of the domain analysis team. The reviewers, who have extensive backgrounds in the domain, will be able to add detail and clarity to the definition. They can also identify areas where the documentation and reference materials are weak and suggest additional materials. The intent of the review is to ensure that the team has identified a coherent domain and obtained resources necessary for analysis.

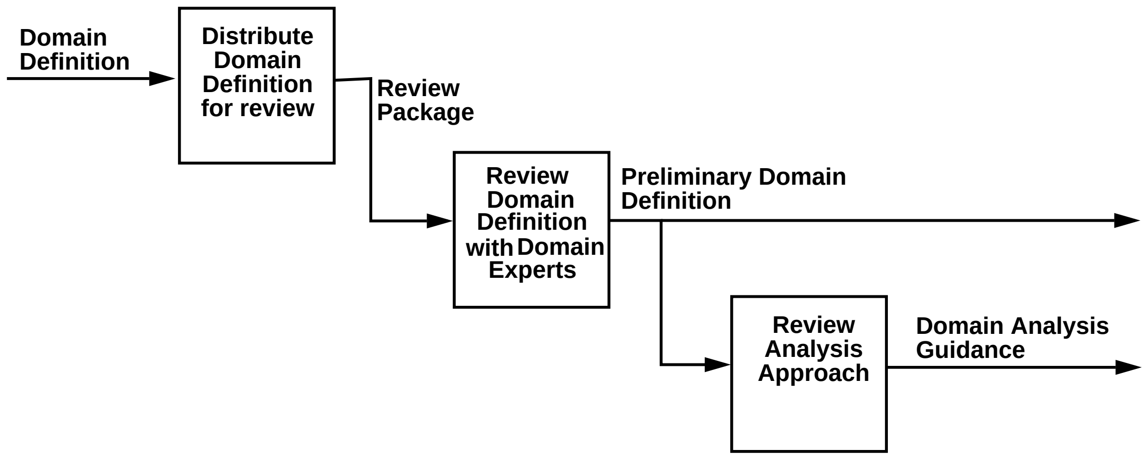

Domain Definition
Distribute Domain Definition for review
Review Package
Review Domain Definition with Domain Experts
Preliminary Domain Definition
Review Analysis Approach
Domain Analysis Guidance

Figure 6-2 Domain Definition Review Process

6.7.1 Distribute Domain Definition for Review

At least two weeks prior to the end of the phase, the domain definition is provided to the reviewers. If the domain experts are not familiar with the CYOOA notation, a meeting may be necessary to teach them the notation.

6.7.2 Review Domain Definition with Domain Experts

In this review, the domain definition is reviewed in detail by the domain experts. The feedback from the review is used to refine the domain definition and to identify additional documentation, reference material, and emerging technology.

The whole-part structures that have been defined by the domain analysis team are presented to the domain experts. These structures need to be examined from a broad perspective to clarify and validate the domain context, its major classes, and their relationships. If an omission from the domain's whole-part structure is identified, a decision to add to the whole-part structure may increase the scope of the domain. It is important that decisions that affect the scope of the domain be made explicitly because what appears to be a small increase in scope can cause a large increase in the domain modeling effort. If structure is added, then the whole-part structure diagram is updated.

The domain definition should be compared with the more abstract systems artifacts. The domain analysis team should examine the designs of existing systems to gain an understanding of how the requirements are satisfied, since the issues which generated a particular implementation may be more apparent in the design than in the requirements. These issues and their rationale are necessary to understand the domain and document the results. In addition, there are usually services or features that have not been documented that are necessary to complete the domain definition. Only by examining designs and code can these features be identified.

The domain definition’s subject diagram and services should be reviewed together, since the domain services define the domain interface. Omissions from the domain services are likely because interfaces may have been overlooked. The users of the domain services can be identified by the reviewers. Walking through realistic domain scenarios will clarify issues.

The domain glossary is discussed to identify definitions that are incorrect, and to ensure that all required terms and concept are defined. If the domain experts can direct the analysis team to standard references from which they may obtain definition and concepts, this will aid in clarifying and documenting the results. Terms and concepts evolve over time within organizations and will represent the bias of the experts. It is important that these terms and concepts be documented using standard means.

Finally, the domain dependencies are examined from the perspective of what services from other systems are necessary for the domain to function. If the domain dependencies are incomplete, then the reviewers can help identify additional dependencies. Any external software dependencies should be identified, examined, and accepted by the reviewers, because a domain will not function without these services.

6.7.3 Review Analysis Approach

Before concluding the review, the analysts will describe their approach to analyzing the domain. The systems that will be analyzed, the breadth of the domain that will be analyzed, and the mix of bottom-up and top-down techniques that will be used are presented to the experts. The experts can recommend adding, deleting, or changing the set of systems to be analyzed. The experts should also be queried to determine what they consider proper coverage of the domain, and how they would approach the analysis proposed by the domain analysis team. Issues that are not resolved are recorded and reported in the review minutes. After the action items, guidance, and suggestions recorded during the review have been discussed, the reviewers should meet without the domain analysis team to determine if they support beginning domain modeling. The reviewers can identify action items for the domain analysis team to complete before they continue. If the reviewers are not satisfied with the approach, the source material, or the domain definition, then any concerns will be reported in the review minutes and discussed with the domain analysis sponsor. It is more cost effective to redirect the analysis early than to try and fix it later.

7 Model Domain (OOA)

In this chapter, the model creation process is described in detail. The process is based on CYOOA and uses the notation of CYOOA to define the domain model. Figure 7-1 gives an overview of the model creation process. The goal of domain analysis is to define a domain model that is derived from an analysis of system artifacts, references, and domain experts so that it can be used to produce RSOs, especially reusable requirements.

The preliminary domain definition and relevant source material are the inputs to the domain modeling phase. Information is extracted from the domain experts to define a top-level view of the domain, while the reference material and domain artifacts will be analyzed to build up the details of the model and increase the team members' understanding of the domain. When a domain expert defines the domain structure, the detailed analysis is faster and easier. The output from this step is the domain definition and model that are defined in detail in Chapter 4.

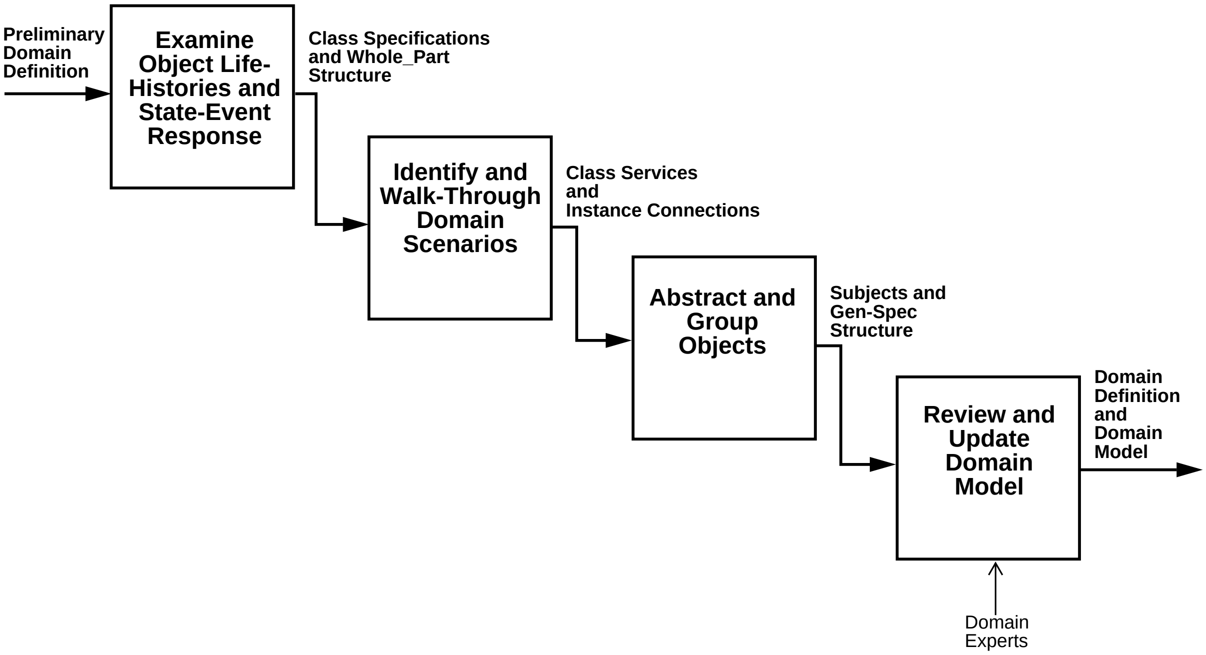

Preliminary Domain Definition
Examine Object Life-Histories and State-Event Response
Class Specifications and Whole_Part Structure
Identify and Walk-Through Domain Scenarios
Class Services and Instance Connections
Abstract and Group Objects
Subjects and Gen-Spec Structure
Review and Update Domain Model
Domain Definition and Domain Model
Domain Experts

Figure 7-1 Domain Modeling Process

The activities for analyzing the domain are defined in CYOOA [7]. Those activities have been evaluated and a further refinement and extension is given below. A specific ordering for the model creation activities is given.

From the domain definition, we get a top-level view of the domain. During the analysis, we will enhance and refine that view by examining the details of the domain. To fully document the domain, a model is created that defines the whole-part structure from the top of the domain down to the lowest level objects. If the analysis team has difficulty defining the high-level structure, they should analyze some lower level objects to get a better understanding. The analysis iterates from the bottom-up to the top-down through the life-history and the state-event response of classes. In general, examining complex examples, when such choices exist, will help identify more of the structure, the services, and the classes. In the beginning, simple examples should be analyzed to ensure that the domain analysis team understands the JODA method and notation. Simple examples are also easier for the team to understand. The simulation of scenarios, which is a later activity, refines and expands the class specifications especially the attributes, services, and instance connections. Concurrently with these activities, the domain analysis team must try to identify generalization and specialization and determine what variations exist in the whole-part structure.

7.1 Examine Object Life-Histories and State-Event Response

There are two techniques defined in CYOOA for identifying the class services, attributes, and instance connections. One technique is to examine the life-history of an object to identify the state changes of each class and identify the services that produce the state changes. This technique does not identify all the error processing and special cases. The second technique is to examine the state-event responses of the class. This information is captured in the class specification using statecharts techniques. For real-time control systems and device driver domains, this information is necessary to define the services.

7.2 Identify and Walk-Through Domain Scenarios

After an initial set of classes has been defined using the techniques above, a process that will increase the precision and completeness of class specifications is to identify and walk through scenarios. These scenarios are documented using a statechart technique and are an extension to the specifications of Coad and Yourdon. Domain experts should identify and initially define the scenarios that are refined, documented, and simulated by the analysis team. These scenarios are documented using statecharts that capture the details. Using the statechart techniques from Statemate, these scenarios can be simulated and validated. The completeness of the analysis will depend on accuracy and realism of the scenarios. To additionally validate the scenarios, they should be reviewed with the domain experts.

7.3 Abstract and Group Objects

From the details that are drawn out through the scenarios, the domain analysis team must raise the level of abstraction. Using the notation of Coad and Yourdon, abstraction is documented with gen-spec structure and subject diagrams. It is through the analysis of the common attributes and services that we identify the generalization-specialization (Gen-Spec) structure that captures the domain variation. For example, an elevator may have Rear Doors, which would require additional Elevator Services Buttons to open and close the Rear Doors.

These additional services are defined in an elevator gen-spec structure. The ability of the notation to accommodate variation is essential for reuse.

Again, simulating scenarios and case studies with the domain experts are the means to identifying the variations within the domain. The CYOOA process does not focus strongly enough on producing or identifying abstract classes in the model. JODA requires examining the domain model to identify abstractions of classes and services. If there are repeating instances of similar problems such as the processing of many message formats, these problems should be analyzed to identify an abstraction which can be instantiated to cover all cases; an example is the Doors object defined in Figure A-6. This abstraction was not identified until after the analysis had been completed, but the abstraction for doors provides an abstraction which represents several domain objects. The domain model was updated to include this abstraction in the domain. The analysis team should attempt to locate and review other domain analyses, to identify abstractions from other domains that apply to the current domain. If the domain being analyzed contains a horizontal domain as defined by McNicholl [21], there may be abstractions in the horizontal domain which help to identify other abstractions.

JODA requires that the domain model be examined to identify stable interfaces where the specifics vary. For example, device drivers for a Graphics Kernel System (GKS) graphics packages have a similar interface, but the implementation can be vastly different because they depend on the capabilities of the device. These instabilities are an important aspect of the domain and require an explicit classification structure to define the different sets of services. To identify these stable interfaces with variable implementations, system documentation and reference material are examined for high-level indicators, like hardware, which imply differing or different capabilities.

A similar problem is identifying any instabilities in the domain. For example, if one can currently use different types of sensors (e.g., infrared, laser, or radar) for targeting, and since the classes, attributes, and services are dependent on the sensor class, then there is some instability in the domain. The domain analysis team should attempt to define a virtual interface with help from the domain experts. This virtual interface would map onto the services for each different member of the class. If a definition of a virtual interface is not possible, the instability and the rationale associated with each variation will be identified to alert potential reusers to the variation. Instability in the domain will not invalidate the domain analysis results since there is value in a partial set of requirements.

7.4 Review and Update Domain Model

Creating and refining the domain model will continue until the reviewers agree that the model is complete and sufficiently accurate to define the domain. Figure 7-2 gives an overview of the review process for the domain model. The purpose of reviewing the model is not only to determine that the model completely and accurately represents and abstracts the knowledge obtained from the documentation, references, and experts, but also to determine that the review obtains guidance and support from the domain experts on how to proceed. The termination conditions for this phase are that the model completely and accurately represents and abstracts the domain knowledge.

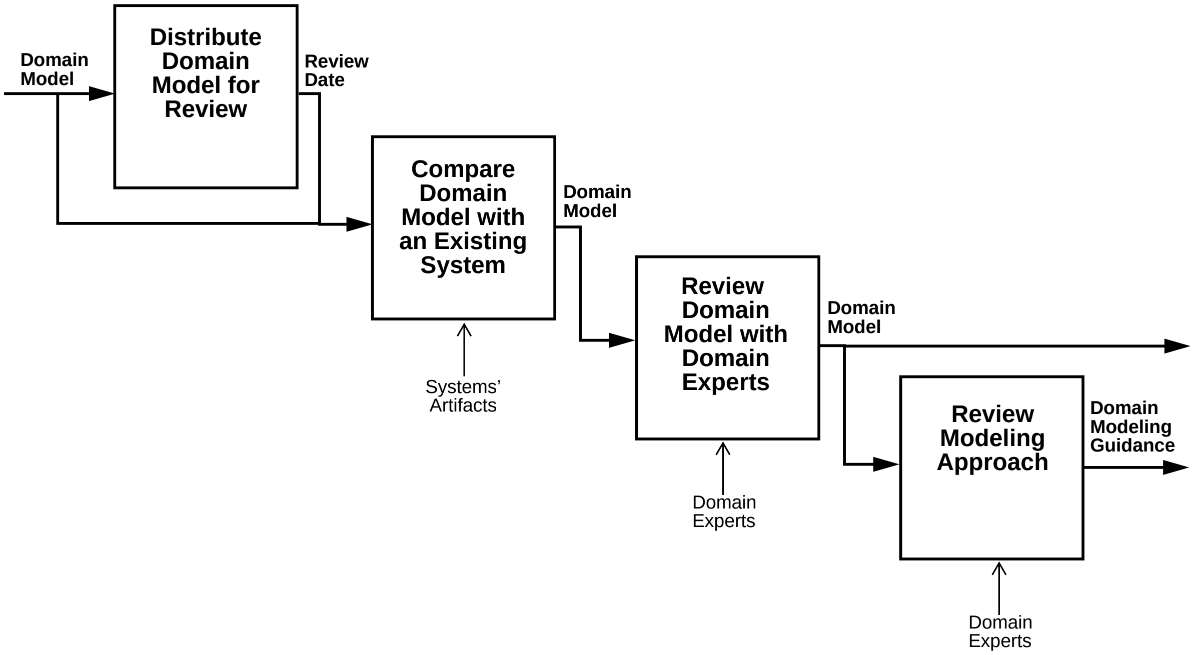


Figure 7-2 Domain Model Review Process

The inputs to the review process are the domain model and the domain definition. The model and definition will be provided to the domain experts along with instructions for performing the review. On-line access to the model may also be provided if desired.

The outputs from the review are the domain expert's guidance, a revised domain definition, and a updated domain model. It is important that this review be constructive so the domain definition and model are improved. The comments of the domain experts are recorded, retained, and later reviewed by the analysis team with the domain experts at the next model review. This does not imply that all comments are accepted, but a valid reason must exist before a domain expert's comment is ignored. The model review is not one review, but a series of reviews which occur during the analysis process. The frequency of the reviews depends on the size of the domain and the length of the analysis. With each review, the completeness and accuracy of the model should increase; if the model is not becoming more complete, accurate, and abstract, then external intervention in the analysis process is required to determine why the analysis is not converging.

JODA recommends that reviewers accept the domain definition and determine if the model represents a complete, accurate, and abstract view of that domain. If the reviewers determine that the model does not adequately cover the domain, then the reviewers should be able provide examples and/or make recommendations for additions to the model. There will be tradeoffs between the scope of the domain definition and the completeness of the model. If, on the other hand, the reviewers don't believe that the domain definition covers the domain, then an evaluation by the domain analysis team is made to determine if the resources exist to change the scope of the domain definition.

7.4.1 Distribute Domain Model for Review

At least two weeks prior to the model review, the domain definition and model are provided to the reviewers. It is important that the review package include both the definition and model because the definition includes the glossary and defines the domain context. If the domain experts are not comfortable with the CYOOA notation, a meeting may be necessary to review the notation. When the experts were identified, they were provided with the notation that will be used so that they may become familiar with it.

7.4.2 Compare Domain Model with an Existing System

One aspect of the review is to request that the reviewers compare the domain model to systems that they have or are currently working with. The differences between the model and other systems should be captured by the domain analysis team for later review and evaluation. In some cases, the current view of the domain may not be entirely compatible with existing systems. For example, ballistic computations have been placed in the mission computer in the F/A-18 because that computer is more powerful, has more available memory, and the Stores Management Computer has exhausted its available memory. The domain analysis team may, however, include the ballistics computation in Stores Management. The reviewers should also be asked to compare the class definitions with objects that would be in that class from their existing systems. It is assumed that any domain expert has information on or is working on systems in the domain.

7.4.3 Review Domain Model with Domain Experts

In addition to the model review material, the domain definition that includes the terms that were compiled during analysis will be given to the reviewers. A major communications problem is that people use the same terms with different definitions. For example, the Common Ada Missile Packages (CAMP) team did not include the guidance functions with their meaning of navigation, but some companies do include guidance with their meanings of navigation. The creation and review of a domain glossary will reduce this problem.

The domain experts are briefed by the domain analysis team on the model. This briefing provides the domain experts with an overview of the model. The domain analysis team will be available before the review to answer question for the domain experts. A certain amount of separation of the domain analysis team from the domain experts is valuable, but domain experts are encouraged to become more involved if they so wish.

The domain experts will have knowledge of systems that have not been included in the analysis. When the domain experts make comments, they should provide, if possible, examples and references. If the domain experts can provide written comments to the domain analysis team before the review, then the domain experts will brief the domain analysis team on any corrections that have been made as a result of the comments.

7.4.4 Review Modeling Approach

Finally, the domain experts will be briefed on how the analysis will proceed from its current state toward the next review. Review materials for the approach are not provided in advance so a detailed explanation is given by the reviewers.

After the domain analysis team has received, reviewed, and updated the model based on the domain expert's comments, they will evaluate the domain definition once again to determine if it is complete, accurate, and appropriately abstract. The domain experts may have increased the scope of the domain definition by recommending the inclusion of classes, services, or structure in the model.

8 Transition to Domain Implementation

There are two issues that haven't been addressed which must be considered before the discussion is concluded. First, the transition to domain implementation must be considered. It is in domain implementation where a generic software architecture is produced that is capable of implementing any of the systems that can be derived from the domain model. Second, the coordination of JODA with the requirements of DoD-STD-2167A must be considered, since all the JIAWG systems listed in Chapter 1 are being acquired using 2167A.

8.1 Transition to Domain Implementation

The development of a generic software architecture which can implement any of the systems defined by the domain definition is the initial goal of domain implementation. The approach recommended is defined by the Parnas [26]. Parnas gives four recommendations for creating such an architecture, and he identifies four indicators that the software architecture is not flexible. The four techniques that Parnas has recommended for making software more adaptable are:

- Requirements definition: identifying the subsets first. He recommends that feasible subsets of operational features be identified. The first step in accomplishing this task is to identify the minimal subset of domain services which makes sense. In the object-oriented case, this implies that some objects may not be in the minimal system. It also implies that there will be a minimal set classes which makes up a useful capability in the domain.
- Information hiding: interface and module definition. The domain model is intended to be general, but a software architecture must be flexible enough to accommodate the extension and contraction. This requires information hiding and module interface definition. One of the strengths of object-oriented programming is that information hiding and module definitions are automatic. Therefore, the natural transition from object-oriented analysis to object-oriented design will support this goal.
- The virtual machine concept. Here, virtual machine is defined in the hardware interface sense not the sense of successive refinement, i.e., interfaces are defined where the software modules provide the instruction set necessary to program the task on a limited set of data types. Another strength of object-oriented programming is that it supports this same view of design. Each class has a set of services from which the problem solution is defined. This technique is especially useful for device drivers or domains whose main task is to control and monitor external hardware.
- Designing the "Uses" structure. In this case, Parnas describes an approach to defining the dependency of one software module on another. This approach [26] is also consistent with the use of classes, but should be specifically included in a method for domain implementation.

The four indicators of non-flexible design identified by Parnas are:

- Excessive information distribution. Assumptions have been that a given feature would or would not be present. When this assumption changes, the design and its implementation are difficult to change.
- A chain of data transforming components. It is often more difficult than it is worth to remove a transformation that is no longer required because it changes the form of the output.
- Components that perform more than one function. Combining two simple features into one component because they are closely related often makes it difficult to extend the system when one functions is required without the other. The separation of function is another reason for desiring an object-oriented approach, because the services are usually separated to begin with, and are easily separated if they are not.
- Loops in the “Uses” relation. If too much interdependency exists between the major elements of a system, then nothing works until almost everything is complete. This makes the ability to subset almost impossible.

The development of object-oriented requirements is consistent with the transition to a flexible design as recommended by Parnas. Thus, the recommended approach will make the next step, the development of a flexible software architecture, easier.

8.2 Relationship of Domain Analysis to DoD-STD-2167A

We have not yet considered the impact of 2167A on the JODA. Since transformation of the domain model into DoD standard requirements produces a specification that is defined by DoD-STD 2167A, then JODA is compatible. The other main products, the domain definition and the domain model, are not deliverables to the customer, and therefore require no specific format. It is important to note that the domain model needs to be maintained so that it can be updated to incorporate new features for each new system. This is essential to maintain the viability of the model over its life time. Without revisiting the domain analysis [17], we cannot obtain the maximum benefits of domain analysis.

Appendix A Office Building Elevator System Diagrams

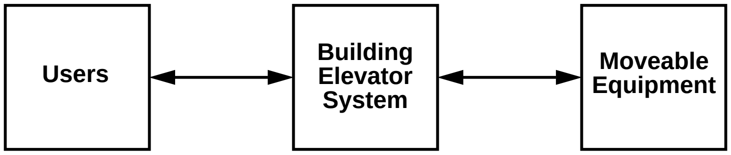


Figure A-1 Top Level Subject Diagram for the Office Building Elevator System

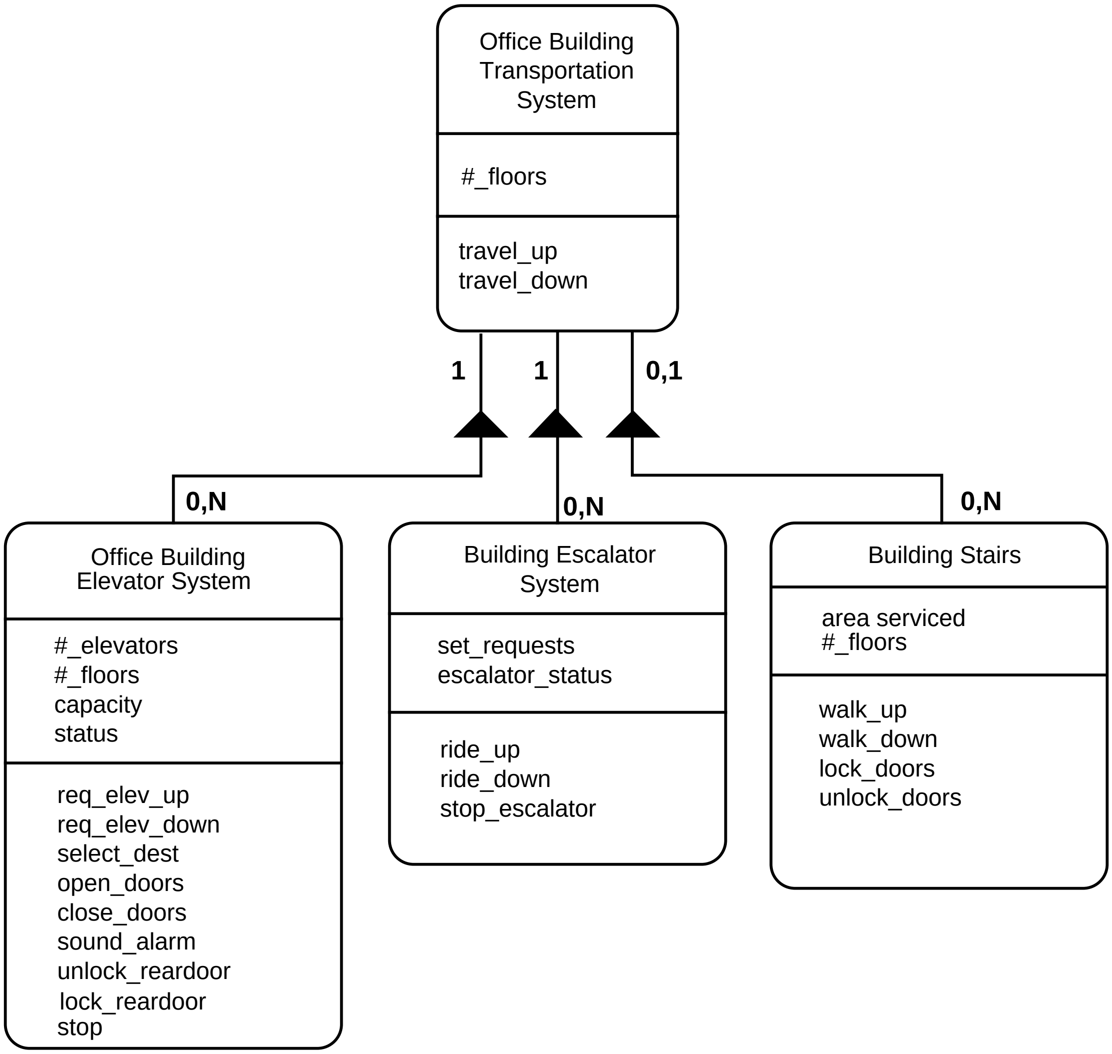


Figure A-2 Assembly Structure Where the Office Building Elevator System is a Part

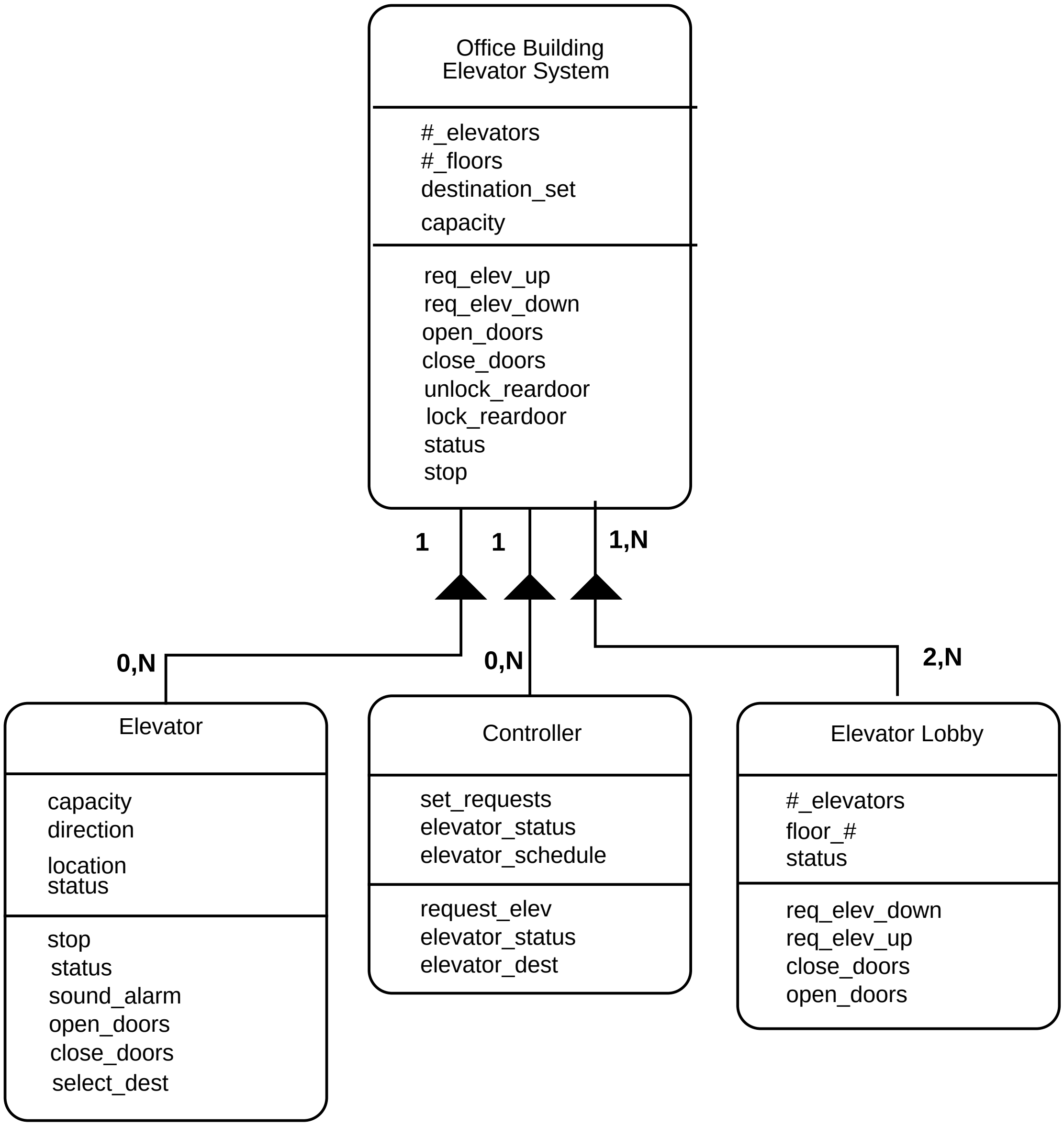


Figure A-3 Top Level Assembly Structure of an Elevator System

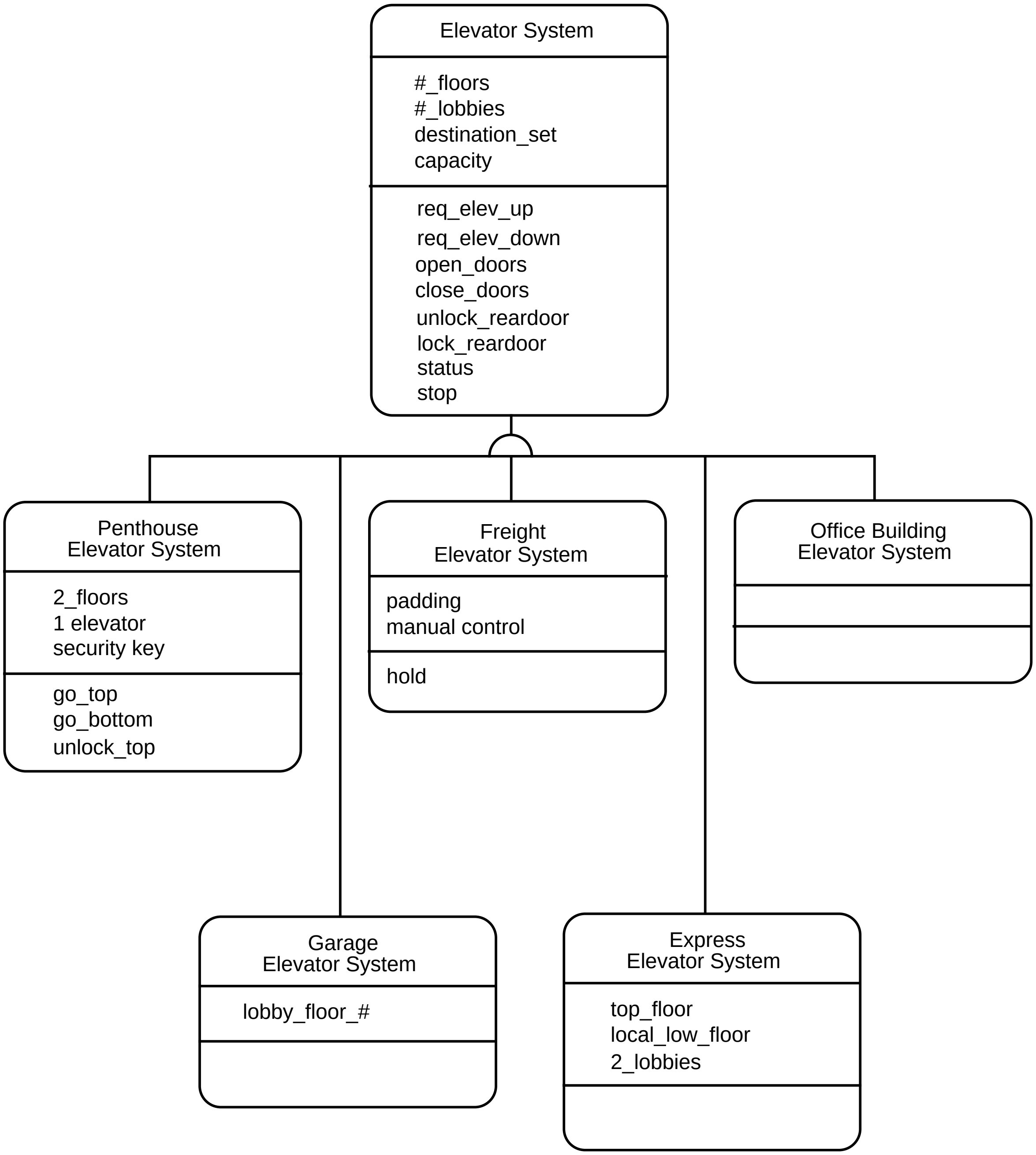


Figure A-4 Top Level Classification Structure of Elevator Systems

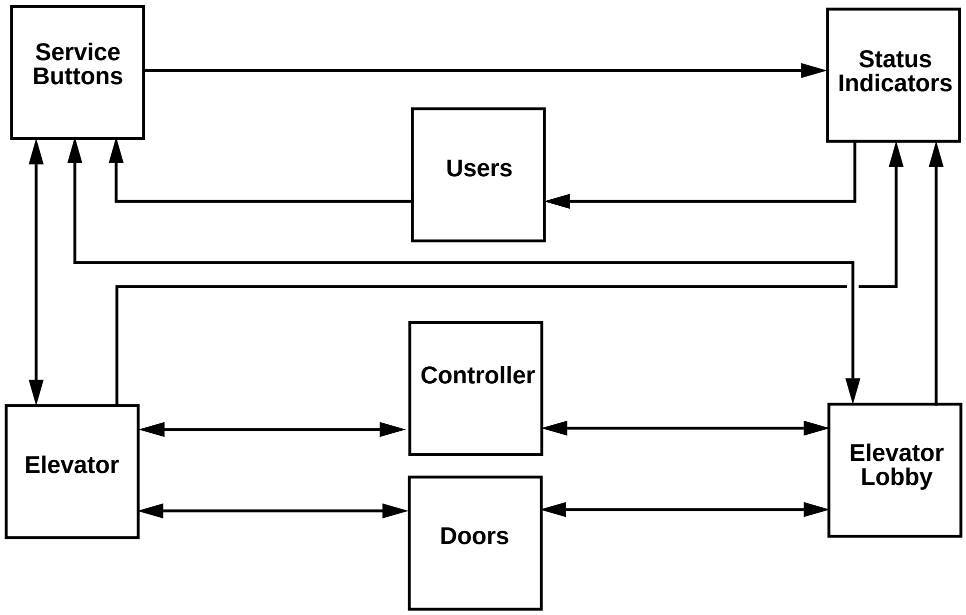


Figure A-5 Second Level Subject Diagram of Elevator System

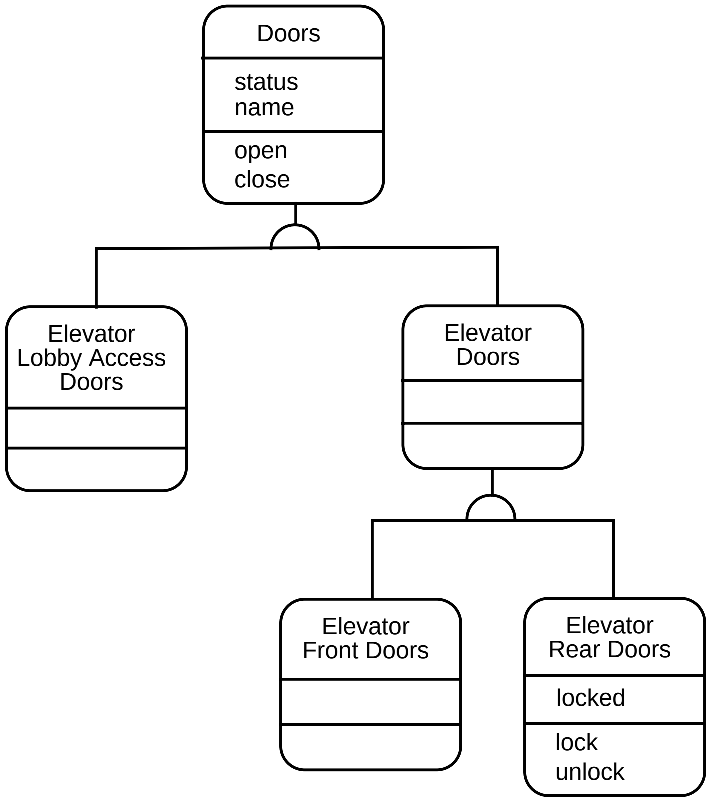


Figure A-6 Door Classification Structures for the Building Elevator System

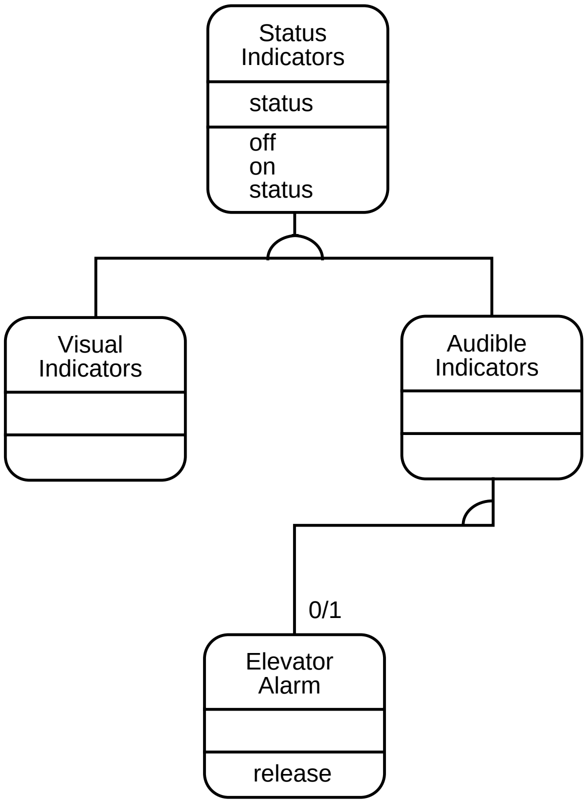


Figure A-7 Status Indicators Classification Structures

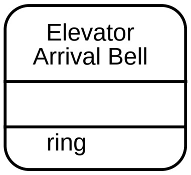


Figure A-8 Elevator Arrival Bell

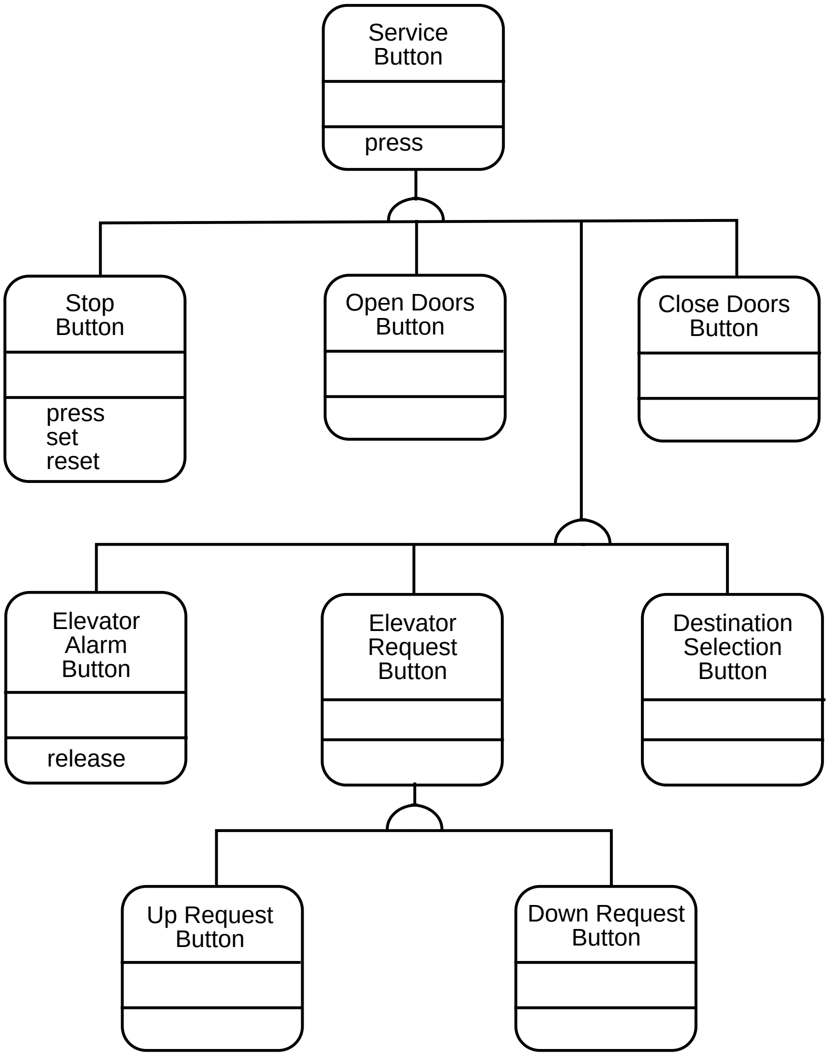


Figure A-9 Service Button Classification Structures

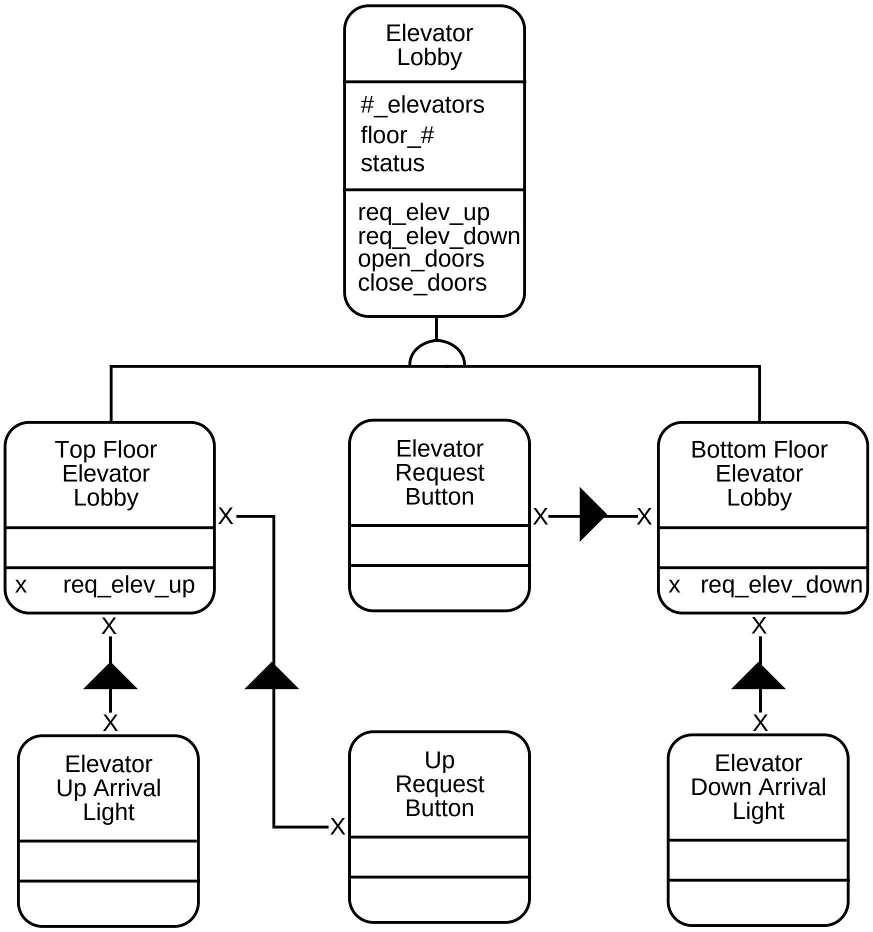


Figure A-10 Elevator Lobby Classification Structures

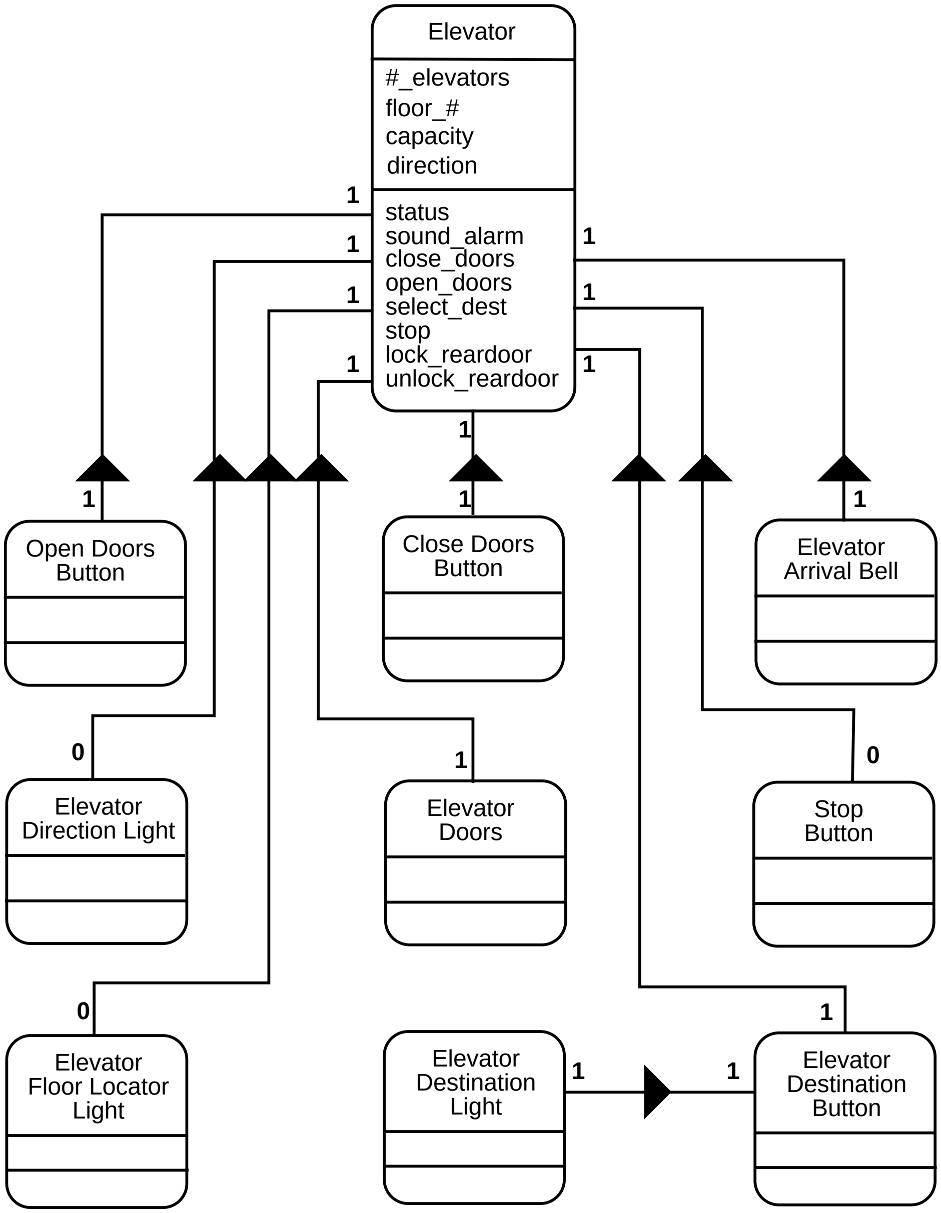


Figure A-11 Elevator Object Diagram for Office Building Elevator

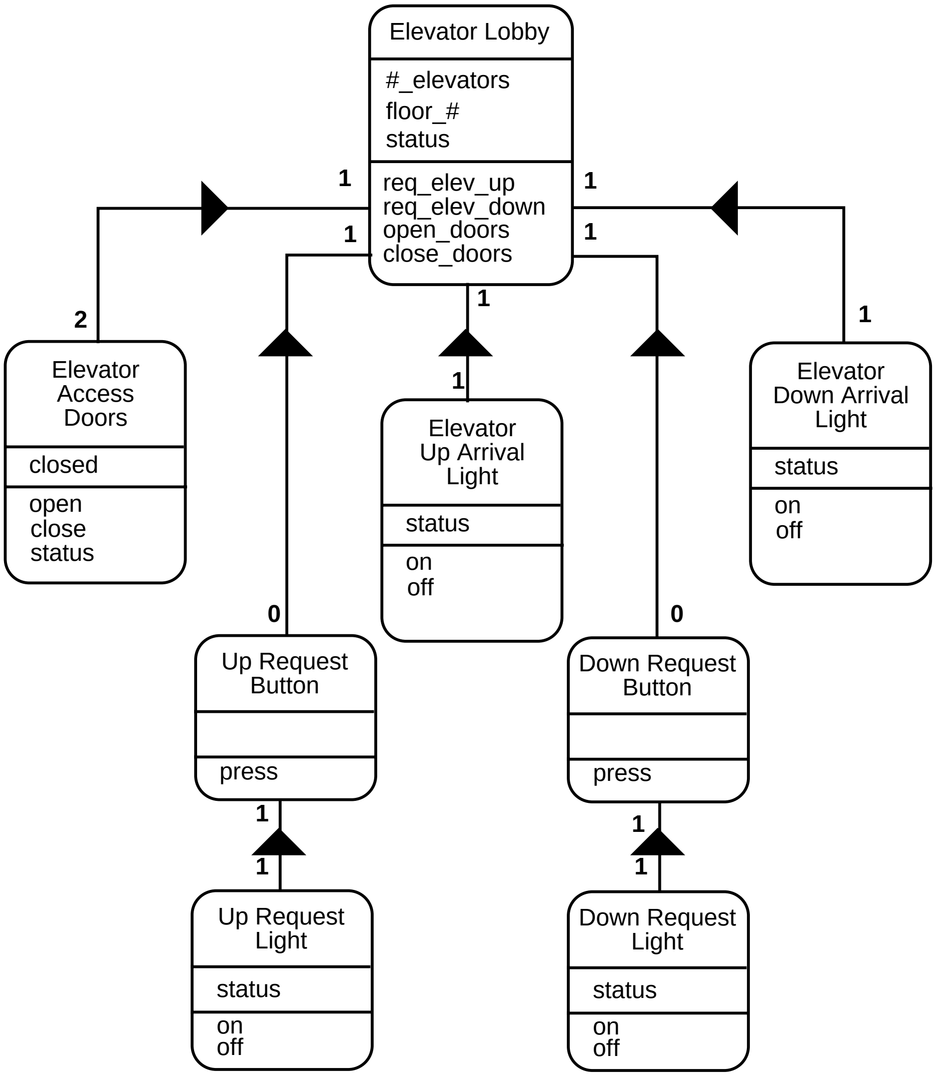


Figure A-12 Elevator Lobby Object Diagram for Office Building Elevator System

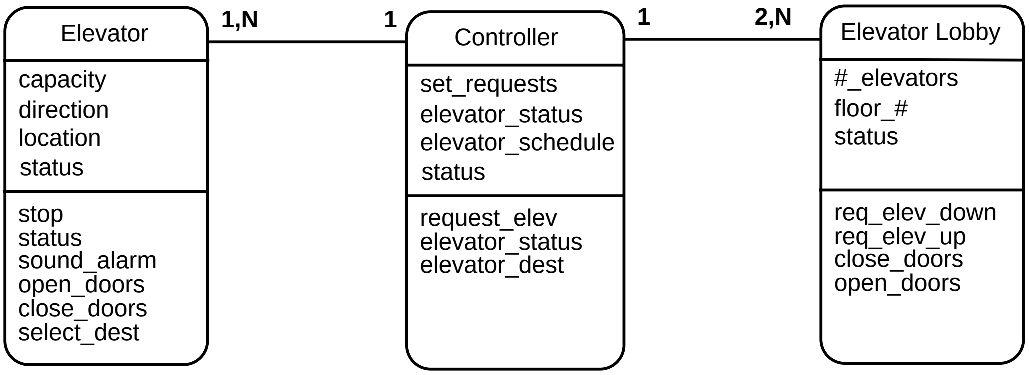


Figure A-13 Controller for the Office Building Elevator System

Appendix B Office Building Elevator System Specifications

B.1 Specification Statement for the Office Building Elevator System

1. There are 1 or more elevators serving 2 or more floors.
2. On board each elevator is a set of destination push buttons, one for each floor, which backlight when depressed, and remain lit until arrival at the selected floor.
3. On board each elevator are two directional signal lights, one for going up, and the other for going down.
4. On board each elevator is a set of lights, on for each floor. One of these lights is always lit, indicating the elevator is at that floor.
5. On each floor there are two summons push buttons, one for summoning the elevator to go up, and the other to go down. These backlight when pushed, and remain lit until an elevator arrives that will go in the selected direction. The top and bottom floors each have only a single summons push button.
6. On each floor, beside each elevator are two floor directional lights, one showing the direction the elevator will take. When an elevator arrives at the floor, the appropriate light show the direction the elevator will take when leaving the floor. The top and bottom floors have only a single directional light each.
7. Each elevator has doors which are either closed or not closed. Opening, closing, or emergency stops are not considered. On each floor, there are doors for each elevator. Both the elevator doors and the floor doors have to be open for people to enter or leave the elevator at a floor.
8. The specification is not concerned with what happens under failure conditions.

B.2 Specification of Elevator Lobby Object of Office Building Elevator System

```
Specification: Elevator Lobby
   definitionData
      #_elevators:  Integer range (1 ... max_#_of_elevators)
                    /*The number of elevator which are accessible from
                      the lobby
      Floor_#:      Integer range (1 ... 102)
                    /*The floor number where the elevator lobby is
                      located.

   externalSystemInput:    Inputs are received from the Up and Down
                           Request Buttons.
   externalSystemOutput:   On and off messages are send to status
                           indicators: Elevator Up Arrival Light,
                           Elevator Down Arrival Light, Down Request
                           Light, and Up Request Light.

    InstanceConnectionConstraint
      with Controller                       1:M, required
      with Elevator Lobby Access Doors      1:M, required
      with Elevator Up Arrival Light        1:M, required
      with Elevator Down Arrival Light      1:M, required
      with Down Request Button              1:M, required
      with Up Request button                1:M, required

   stateEventResponse:     When Users press UP or Down Request Buttons,
                           these requests are sent to the controller
                           unless a previous request has been sent.

   intent/purpose:         Provides a locus of control for elevator
                           doors, requests, and status indicators.

   Service request_elevator_up          Data flow: None.
      Sends request_elevator message to controller, after validation

   Service request_elevator_down        Data flow: None.
      Sends request_elevator message to controller, after validation

   Service close_access_doors           Data flow: None.
      Sends close_doors message to Elevator Lobby Access Doors
   Service open_access_doors            Data flow: None.
      Sends open_doors message to Elevator Lobby Access Doors

   end specification
```

B.3 Elevator Specification for Office Building Elevator System

```
Specification: Elevator

  definitionData
     floor_#:  integer range (1 ... #_floors)
               /*Remembers the floor on which the elevator is located
     capacity: integer range (0 ... 50,000)
               /*The weight or number of people the elevator can carry
     direction:Enumerated (up, down)
               /*Remembers the direction of travel of the elevator.
     destinations:set of integers
               /*Remembers what floors the elevator will stop on.

  externalSystemInput     Users may request service by pressing or
                          activating any of the Elevators service
                          buttons.

  externalSystemOutput    Elevators status indicators are activated
                          to acknowledge User requests and to signal
                          events the User should be aware of.

  InstanceConnectionConstraint
     with Open Doors Button                1:1, required
     with Close Doors Button               1:1, required
     with Elevator Arrival Bell            0:1, optional
     with Elevator Direction Light         0:2, optional
     with Elevator Doors                   1:2, required
     with Stop Button                      1:1, required
     with Elevator Destination Button      1:M, required

  stateEventResponse      The elevator takes Users requests,
                          acknowledges them, forwards them to the
                          controller, and takes the actions directed
                          by the controller.

  objectLifeHistory

  intent/purpose          The purpose of the Elevator is to move the
                          Users from floor to floor. The Elevator does
                          not control its movements, this is normally
                          done by the controller.

  Service status                   Data flow: None.
     When requested by the controller, or when the status of the
     elevator changes, the Elevator sends its status to the
     controller.
```

```
Service sound_alarm              Data flow: None.
   When the User presses the Alarm Button, the Elevator turns on
   the alarm. When the User releases the Alarm Button, the Elevator
   turns off the alarm.

Service close_doors                Data flow: None.
   When the Elevator Doors are open, and either the Users presses
   the Close Door Button or the controller directs the elevator to
   close the doors, the doors are closed. This action is
   synchronized by the Elevator Lobby.

Service open_doors                 Data flow: None.
   When the elevator is located at a floor, and either the User
   presses the Open Door Button, or the controller directs the
   elevator to open the doors, the doors are opened. This action is
   synchronized by the Elevator Lobby.
Service select_dest                Data flow: None.
   The Users presses one of the Elevator Destination Buttons and
   the request is sent to the controller, after ensuring that the
   request is not currently active.

Service Stop                       Data Flow: None.
   When the users activates the stop button, the elevator is
   stopped, and remains stopped until the User releases the stop
   button.

end specification
```

B.4 Controller Specification for Office Building Elevator System

```
Specification: Controller

   definitionData
      set_requests: set of records, record contains: type_of_req,
                orig_of_request, destination
                /*The controller remembers each request it receives
                until that request is handled by adding a stop to an
                elevator_schedule
      elevator_schedule: set of records, record contains: elevator_#,
                location, elevator_destinations
                /*The controller remembers on what floor the elevator
                each elevator is located, and where it will stop.

   alwaysDerivableAttribute
      elevator_status: set of records, record contains: elevator_#,
                location, direction
                /*The controller keeps the status of all the elevators.
                The status the controller needs for each elevator is
                the location (floor #), and its direction of travel.

   InstanceConnectionConstraint
      with Elevator                        1:M, required
      with Elevator Lobby                  1:M, required

   objectLifeHistory    The controller receives request, and assigns
                        them to elevators.

   intent/purpose       The controller manages the assets of the OBE
                        system, the elevators. It provides fair and
                        equitable service to the Users.

   Service    request_elev          Data Flow: None.
      This request is received from an Elevator Lobby. The request is
      added to the set_requests and serviced in a FIFO order.

   Service    elevator_status       Data Flow: None.
      This message is received from an Elevator to update its status
      and elevator_schedule.

   Service    elevator_dest:        Data Flow: None.
      elevator_dest is a message received from an Elevator to request
      that it stop at a given floor. The request is validated against
      the elevator_schedule, assigned based on the elevators schedule
      and direction.

   end specification
```

B.5 Service Button Specification

```
Specification: Service Button

  externalSystemInput    Users press the button to request service.

  externalSystemOutput   The associated light is lit to acknowledge
                         the user’s request.

  InstanceConnectionConstraint
     with Elevator Lobby              1:1, required
     with Elevator                    1:1, required
     with Status Indicator            0:1, optional

  stateEventResponse     Users presses a Service Button, light is
                         turned on, and request is sent for service.

  objectLifeHistory

  intent/purpose         Provide the Users with means to access
                         system services.

  Service Press                      Data flow: None.
     If a previous request has not been satisfied, illuminate the
     associated Status Indicator, if present, and request service.

  end specification
```

B.6 Open Doors Button Specification

```
Specification: Open Doors Button

  externalSystemInput    Users press button to open Elevator Doors.

  InstanceConnectionConstraint
     with Elevator                    1:1, required
     is-a                             Service Button

  stateEventResponse     Users presses a the button, and request is
                         sent for service.

  intent/purpose         Provide the Users with means to open the
                         Elevator Doors.

  Service Press                      Data flow: None.
     If the Elevator doors are not already open a request is send to
     the Elevator to open the doors. Elevator determines if the doors
     should be opened.

  end specification
```

B.7 Status Indicator Specification

```
Specification: Status Indicator

   definitionData:
      status:   logical range (true, false)
                /*Remembers whether the indicator is on or off.

   externalSystemOutput   Light is illuminated to acknowledge a User
                          request or to signal an event the User
                          should be aware of.

   InstanceConnectionConstraint
      with Service Button                0:1, optional
      with Elevator Lobby                0:1, optional
      with Elevator                      0:1, optional

   objectLifeHistory      Indicator is turned on and off as directed
                          through its instance connections.

   intent/purpose         To notify the User of status, either to
                          acknowledge a User request or to signal an
                          event.

   Service On                       Data flow: None.
      Turn on the indicator.

   Service Off                      Data flow: None.
      Turn off the indicator

   Service Status                   Data flow: None.
      Returns whether or not the indicator is on.

   end specification
```

B.8 Elevator Alarm Specification

```
Specification: Elevator Alarm

externalSystemOutput   Sound is generated to alert other building
                       occupants to problems with an elevator.

InstanceConnectionConstraint
   with Elevator Alarm Button         1:1, required.

intent/purpose         The means for the User to signal a problem.

Service On/Off                     Data flow: None.
   Turn on/off the alarm.

end specification
```

B.9 Door Specification for Office Building Elevator System

```
Specification: Doors

descriptiveAttribute
   Name:     String range (1 ... 32) characters
             /*used to distinguish between different doors

definitionData
   status:   enumerate range (open, closed)
             /*Maintains if the doors are open or closed.

externalSystemOutput   When directed the Doors are open and closed.

InstanceConnectionConstraint
   with Elevator Lobby                1:1, required
   with Elevator                      1:1, required

stateEventResponse     When User presses Open/Close Doors Button,
                       or when directed, Doors are opened/closed.

intent/purpose         The Doors control access to the elevators.

Service open/close                 Data flow: None.
   When directed, the Doors are opened/closed.
Service status                     Data flow: None.
   When requested or when the status has changed, this message is
   sent to the Elevator or Elevator Lobby as appropriate.

end specification
```

B.10 Elevator Lobby Access Door Specification

```
Specification: Elevator Lobby Access Doors

   descriptiveAttribute
      Name:     String range (1 ... 32) characters
                /*used to distinguish between different doors

   definitionData
      status:   enumerate range (open, closed)
                /*Maintains if the doors are open or closed.

   externalSystemOutput   When directed by Elevator Lobby the Doors
                          are open and closed.

   InstanceConnectionConstraint
      with Elevator Lobby                 1:1, required

   stateEventResponse     When the Elevator arrives at the floor, the
                          controller directs both the Elevator and
                          the Elevator Lobby to open the doors
                          associated with the Elevator. When the
                          Elevator Lobby Access Doors in the lobby
                          sense, the Elevator Doors opening, the
                          Elevator Lobby Access Doors also open.

   intent/purpose         The Doors control access to the elevators.

   Service open           Data flow: None.
      When directed, the Doors are opened.


   Service close          Data flow: None.
      When directed, the Doors are closed.


   Service status         Data flow: None.
      When requested or when the status has changed, this message is
      sent to the Elevator Lobby.

   end specification
```

B.11 Elevator Arrival Bell Specification

```
Specification: Elevator Arrival Bell

  externalSystemOutput   Sound is made to alert the User.

  InstanceConnectionConstraint
    with Elevator                      1:1, required

  objectLifeHistory      The bell is used by the Elevator only.

  intent/purpose         To notify the User that the Elevator has
                         arrived at a floor where the doors will
                         open.

  Service ring                        Data flow: None.
    When requested by the Elevator the bell is rung to signal arrival
    at a floor.

  end specification
```

B.12 Object Specification Template

```
Specification: <Name of Object>

  descriptiveAttribute
     <Name>:   <Type> range (<Low> ... <High>)
               /*English description

  definitionData
     <Name>:   <Type> range (<Low> ... <High>)
               /*English description

  alwaysDerivableAttribute
     None.

  occasionallyDerivableAttribute
     None.

  externalSystemInput
     None.

  externalSystemOutput
     None.

  InstanceConnectionConstraint
     with <Name>                        <0 | 1>:<1 | M>,
                                        <required | optional>

  stateEventResponse

  objectLifeHistory

  notes

  intent/purpose

  Service <Name>         Data flow: None.
     <English description>

  end specification
```

Appendix C CYOOA Notation and Process

In this Appendix, we define the CYOOA process and notation that are used throughout the document to define the process and products of object-oriented domain analysis. The purpose in describing the notation is to:

- define the notation which is used in the domain definition and the domain knowledge base, and
- define the activities that are used in object-oriented analysis.

C.1 CYOOA Notation

As stated previously, the domain analysis method presented here is based on *Object-Oriented Analysis* [7] by Coad and Yourdon and uses the notation from that book. We have not determined whether or not this notation is adequate to cover all situations, and in general, Coad and Yourdon recommend using whatever is needed. Figure C-1, Figure C-2, Figure C-3, Figure C-4, and Figure C-5 describe much of the notation used by Coad and Yourdon. The example presented in Appendix A uses this notation. Furthermore, we use this notation when describing the domain analysis process.

C.1.1 Class/Object Notation

Figure C-1 gives the generic definition of a class/object. The class/object name goes in the top section. All class/object names will begin with capital letters. For example, the top object in Figure C-2 describing an elevator lobby is “Elevator Lobby”. This use of lower case will distinguish between descriptive terms and class/object names. The attributes associated with the class/object are given in the middle box. There may be as many attributes as needed. Finally, the names of the services are given in the bottom section. Once again, there maybe as many services as needed. For a large number of attributes or services consider using two columns.

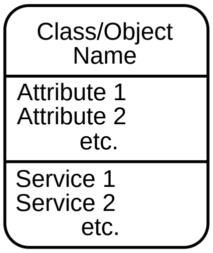


Figure C-1 Class/Object Notation

C.1.2 Generalization-Specialization Notation

Figure C-2 is an example of a Generalization-Specialization diagram (also referred to as a “Gen-Spec Diagram” and “Gen-Spec Structure.”) (denoted by the “O” at the junction of lines below the Elevator Lobby Object) defined by Coad and Yourdon. Figure C-2 defines the Gen-Spec structure for the class Elevator Lobby. In addition to the class Elevator Lobby, there are two objects: Top Floor Elevator Lobby, and Bottom Floor Elevator Lobby. In the class definition for Elevator Lobby, there are four attributes:

#_elevators The number of elevators which stop at this floor.

floor_# The floor number for the elevator lobby.

up_elev_arr The status of whether an elevator has stopped at this floor and will discharge or take-on passengers before continuing up.

down_elev_arr The status of whether an elevator has stopped at this floor and will discharge or take-on passengers before continuing down.

This description identifies two services performed by the elevator lobby:

up_elev_req User requests that an elevator stop at this floor to transport them up.

down_elev_req User requests an elevator stop at this floor to transport them down.

In the elevator lobby example in Figure C-2, two specializations of elevator lobby have been identified: Top Floor Elevator Lobby, and Bottom Floor Elevator Lobby. When an “x” appears before an attribute or service in an class definition, the “x” means that this class does not possess that attribute or service.

The presence of a “return” service in the Bottom Floor Elevator Lobby is a service the building management can set to cause elevators to return to the bottom floor when not in use. This service is only available in this instance/subclass.

This is not a complete description of the options or notation for Gen-Spec structure; please examine Chapter 4 of *Object-Oriented Analysis* [7]. For a more complete description of attributes and services read chapters 6 and 7 respectively of *Object-Oriented Analysis* [7].

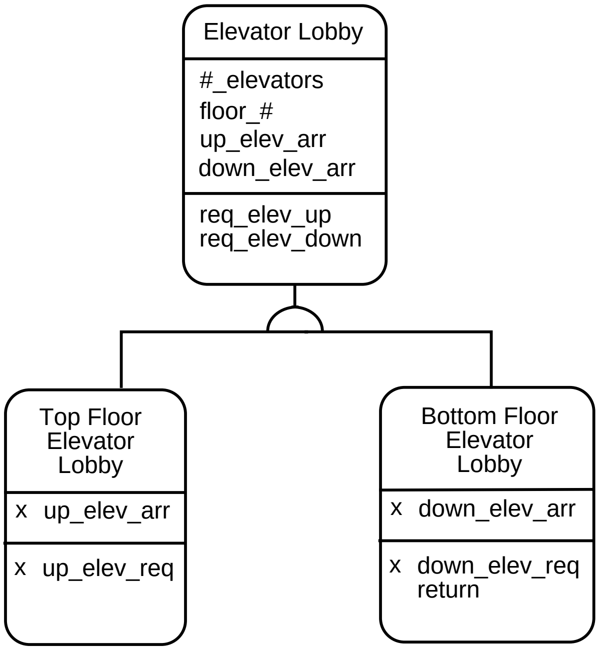


Figure C-2 Gen-Spec Diagram

C.1.3 Whole-Part Notation

Figure C-4 is an example of Coad and Yourdon’s whole-part diagram (also referred to as a “whole-part diagram” or “whole-part structure”). The triangles below the Elevator Lobby class indicate that this is a whole-part diagram, and the triangle points from the part to the whole. This is similar to an ER Diagram where the relationship between the higher class and the lower object is one of the whole to its parts. Figure C-4 shows that an Elevator is composed of Elevator Access Doors, Service Buttons, and Status Indicators. Service Buttons and Status Indicators are examples of another class. In Figure C-4, the specific service buttons and status indicators that are part of an Elevator Lobby have not been defined. From the relation connecting Elevator Lobby to Elevator Access Doors, each Elevator Access Door is defined to be associated with one and only one Elevator Lobby. This constraint is defined by the **“1”** nearest Elevator Lobby on that relation.

At the other end of that relation, the **“1,N”** indicates there can be multiple Elevator Access Doors in an single Elevator Lobby, but there must be at least **“1”**, Elevator Access Door associated with each Elevator Lobby. Also, please note that there is a relation between Service Buttons and Status Indicators. The **“0,1”**’s on this relation identify this as an option. That means that we can have Service Buttons defined with and without associated Status Indicators. Examining the relation in the other direction, we see that each Status Indicator may be associate with or without Service Button, but each Status Indicator is associated with at most one Service Button.

Examining this notation, we see that there are four possibilities:

- Either 0 or1 object B associated with each object A.
- between 0 and N object B’s associated with each object A.
- one and only object B associated with each object A.
- between 1 and N object B’s associated with each object A.

Once again, this is not a full description of either Whole-part Notation or instance connections (relations). For a more comprehensive treatment read Chapters 4 and 6 of *Object-Oriented Analysis* [7].

C.1.4 Subject Diagrams

Subject diagrams are used to group classes and subjects into higher level views. Figure C-5 is a very simple subject diagram, which provides a slightly higher view of the interaction between Users, Service Buttons, and Status Indicators. The arrows are unidirectional, i.e., the arrow which points from Status Indicators to User means that a message is sent from the Status Indicator to the User. The User sends a message to the Service Button (the message is really the “Press” service), and finally, Service Buttons send messages (like On/Off) to Status Indicators. It is essential to have a grouping mechanism to control the amount of information that the users are exposed to at one time. Another example is the second level decomposition of the Elevator System given in Appendix A, Figure A-5.

0,1 A------------B	In the relation between class A to class B, there are between 0 and 1 instances of class B associated with each instance of class A.
0,N A------------B	In the relation between class A to class B, there are between 0 and n > 0 instances of class B associated with each instance of class A.
1 A------------B	In the relation between class A to class B, there is 1 and only 1 instance of class B associated with each instance of class A.
1,N A-------------B	In the relation between class A to class B, there are between 1 and n > 0 instances of class B associated with each instances of class A.

Figure C-3 Instance Connection Constraint Notion

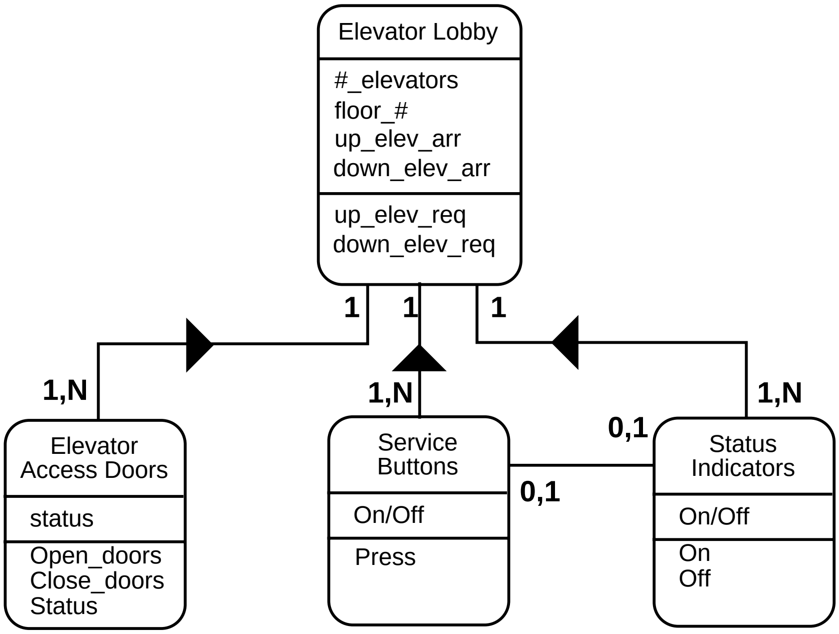


Figure C-4 Whole-Part Notation

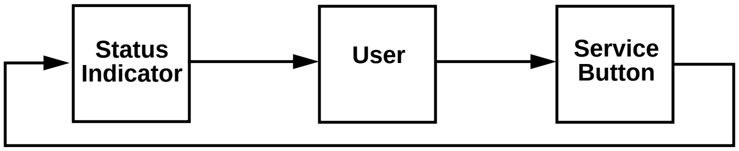


Figure C-5 Subject Notation

C.2 Analysis Activities

Object-Oriented Analysis [7] defines five activities for the analysis process. They do not prescribe a specific ordering for those activities, but the analysis moves from one activity to another and back very freely. The steps of these activities below are outlined below, but the order is not prescribed. In Chapter 4, we describe an approach to applying these activities.

C.2.1 Identify Classes

A Class is an encapsulation and an abstraction: an encapsulation of attributes and exclusive services on those attributes and an abstraction of the problem space, representing one or more occurrences of something in the problem space. To find Classes, look at problem space, text, and pictures (previous systems).

What to look for:

1. Gen-Spec and Whole-part structure (see C.2.2).
2. Other Systems that the system under development (SUD) will interact with.
3. Devices (documents, formal) that the SUD will interact with.
4. Events that must be remembered by the SUD.
5. Roles played by humans who interact with the SUD and about whom information is kept by the system.
6. Locations (physical), offices, or sites that the system under consideration will need knowledge of.
7. Organizational units that humans belong to.

What to consider:

1. Does the SUD need to remember anything about this potential Class?
2. Does the system need to provide processing on behalf of this Class?
3. Does the Class have more than one attribute?
4. Are there attributes which apply to all instances of an Class?
5. Are there a common set of services for all instances?
6. Essential requirements - Are there requirements that system must have, regardless of the technology used for implementation?

What to challenge:

1. Unneeded remembrance - If a system does not need to hold information about a real world thing over time or provide Services for it, remove the Class.
2. Unneeded Services - If a system does not need to hold information about a real world thing over time or provide Services for it, remove the Class.
3. Single occurrences - If there is only one occurrence of a Class try to identify Classes with the same attributes and services and just use one of them.
4. Derived results - Examine the model for derived results, e.g., customer’s age in a system that already remembers the customer’s date of birth.

How to name:

1. Use a singular noun, or adjective+noun
2. Use a name that describes a single instance of a Class
3. Use standard subject matter vocabulary - match the names to the domain expert's vocabulary.
4. Use readable names, with upper and lower case; don't use prefix and suffixes.

C.2.2 Identify Structure

Structures represent complexity in the problem space:

1. Generalization-Specialization (Gen-Spec) Structure represents class-member organization, reflecting generalization-specialization.
2. Whole-Part Structure represents aggregation, reflecting whole and component parts.

How to define:

1. Gen-Spec Structure: Consider each Class as a generalization, then as a specialization. Does the specialization reflect meaningful real-world specialization, and is it within the problem space?
2. Whole-part Structure: Consider each Class as a whole, then as a component part. Examine the next higher level Class to check problem space and scope.

   Checks: Is this real-world structure, within the problem space, and within the scope of the domain?

C.2.3 Identifying Subjects

This is a mechanism for controlling how much of a model a reader considers at one time.

How to define:

1. Add a subject for each Class.
2. Add a Subject for each structure.
3. Group Subjects if more than 7 +- 2.
4. Combine tightly coupled subjects to provide a better overview after attributes and services are defined.
5. For very large projects, iterate quickly over Classes, Structure, and Subjects.

Show communications between Subjects with connections, and number the Subjects.

C.2.4 Defining Attributes

Attributes are data elements used to describe an instance of a Class or Gen-Spec Structure.

How to define Attributes:

1. Return to the problem space description and the user to get the values (state) of the domain objects.
2. Insure attributes really model reality.
3. Identify each Attribute at the atomic level, e.g. address, not street, city, state, and zip code.

Position the Attributes:

1. Place the attribute using inheritance in Gen-Spec Structures.
2. Place at the highest level where it applies to most specializations.
3. Add specializations when they only exist in the specialization.

Identify and Define instance connections:

1. Add instance connection lines between Classes to identify problem space connections (relationships).
2. Add instances to specializations if the instance doesn't apply to ALL specializations.
3. Define multiplicity, i.e. define whether the instance is 1:1 or 1:m for both ends of the instance. Add a **"1"** for 1-1; add "N" for multiples.
4. Define participation, is the instance connection mandatory or optional for each end of the connection. Add **"1"** for required; **"0"** for optional.
5. Check for special cases
    - a. Connections across three or more Classes. Add an instance of A to C, if A to B or B to C is optional.
    - b. Many-to-many instance Connections. Examine the connection to determine if the attributes describe the connection between the Classes. If so, then add a Class of things remembered.
    - c. Instance Connections between instances of the same Class or Gen-Spec Structure. If the instance connection has descriptive Attributes, then introduce a new Class to capture the Instance Connection.
    - d. Multiple Instance Connections between two Classes or Gen-Spec Structures. Examine the meaning of the two connections, capture the underlying semantic distinction with one or more Attributes, and remove one instance.

Class and connection identifiers are defined by the system during design phase since real-world identifiers will repeat.

Revise Classes:

1. Attributes with value of “Not Applicable”. If some Attributes do not apply to all instances, then consider adding more Gen-Spec Structure.
2. Single Attributes. If a Class or an instance of a Gen-Spec Structure has only a single Attribute, then the model can be revised to reflect a higher level of abstraction.
3. Repeated values for one or more Attributes. If an attribute can simultaneously have more than value, then consider adding a new Class.
4. Adaptation Parameters. For site specific data, add a site Class with installation Attributes. For operational data, parameterize the range Attributes.

Specify the Attributes:

1. For each Class or structure, specify the Attributes with names and descriptions.
2. For each Class or Structure, identify each category of attribute (e.g., descriptive, definition, always derivable, occasionally derivable).

Specify instance connection constraints. For each Class and structure, specify the instance connection constraints.

C.2.5 Defining Services

Services are the processing to be performed upon receipt of a message.

How to define Services:

1. Identify the Services (Primary Strategy):
    a. Occur (instance add, change, delete, and select)
    b. Calculate Services - calculate results for an instance or on behalf of another instance.
    c. Monitor Services - performs on-going monitor of an external system, device, or user.
2. Identify the Services (Secondary Strategies)
    a. Object Life History:
        i. Define the basic Object Life History sequence.
        ii. Check for variations on each step.
        iii. Add to the basic sequence.
        iv. Add Services.
    b. State-Event Response:
        i. Define major system states.

ii. For each state, list the external events and required responses.

iii. Expand the Services (and Message Connections).

3. Identify the Message Connections. Begin by adding Message Connections between Classes and Gen-Spec Structures already connected with Instance connections. Arrows are unidirectional (meaning send, receive, and response).

4. Specify the Services

   a. Focus on required externally observable behavior. Can this requirement be externally observed (tested)? Shall --> observable when the system is tested, Will --> something that will happen, but it is not under the control of the system, Present tense --> all others.

   b. Use a template. Specify the occur, state-event response, etcetera, with structured English to include exceptions.

   c. Add Diagrams to simplify Service specifications. Use data flow diagrams, statecharts, etcetera.

   d. Add supporting tables. To summarize the interactions between Classes, especially for real-time systems, utilize:

      i. A summary of Services and applicable states

      ii. Threads of execution analysis

      iii. Timing and sizing budgets

   e. Develop Service Narratives--If you must.

      i. Services specified with bullet lists keep the textual specification concise and well-focused.

      ii. If narrative text is required and/or desired, get a technical writer to translate.

   f. Put the documentation set together. The full package contains:

      i. OOA Diagrams: Subject, Class, Structure, Attribute, Services Layers

      ii. OOA Repository (one entry per Service or Gen-Spec Structure)

      iii. Supporting Tables (if any), Services and Applicable States Table, Critical Threads of Execution Table, Timing and Sizing Table.

Glossary

We would prefer to have more commonly accepted definitions, but the process of domain analysis is insufficiently mature for this agreement.

attribute	a data element used to describe an instance of an object or classification structure
application engineering	the process of analyzing, specifying, designing, implementing, integrating, and testing software (systems) based on the existence of a reuse-based software engineering methodology, domain knowledge, domain assets, and reuse tools
class	a collection of objects which can be described with the same attributes and services
domain	1. a territory or range of rule or control: realm; 2. a sphere of concern or function; field: the domain of history
domain analysis	the process of identifying, collecting, organizing, analyzing, and representing important entities in a domain model from the study of existing systems, underlying theory, emerging technology, and development histories within the domain of interest
domain engineering	the process of identifying candidate domains from a cost performance analysis, selecting domain analysis, design, and implementation methods, performing domain analysis, domain design, and domain implementation, and creating domain assets to support Reuse-Based Software Engineering (Application Engineering)
domain entity	an object, relationship, operator, process, fact or rule of a domain [14]
domain implementation	1. the construction of a software architecture, components, methods, and tools and their supporting documentation to solve the problems of system/subsystem development by the application of the knowledge in the domain model; 2. the process of building reusable software objects using domain analysis work products (domain models, domain languages, and taxonomies); 3. the creation of domain language processors and allied tools
domain model	a formal, concise, representation of a domain [14]
entity	1. the fact of existence; being; 2. something that exists independently, not relative to other things; 3. a particular and discrete unit; an entirety
gen-spec diagram	a representation of class-member organization, reflecting generalization-specialization
model/requirements transformation	the process of converting the domain model into requirements that are defined by DOD-STD 2167A Data Item Description DI-MCCR 80025A

object

an abstraction of something in the real world, reflecting the capabilities of a system to keep information about it, interact with it, or both; an encapsulation of attribute values and their exclusive services.

reusable software object

Life-cycle products that are created during the software development process that are needed to operate, maintain, and upgrade deliverable avionics during its lifetime and that have the potential for reuse. The objects may include, but are not limited to: requirements specifications, design documents (both top level and detailed), source and object code, test specifications, test code, test support data, users manuals, programmer notes and algorithms.

service

a processing to be performed upon receipt of a message by an object

software architecture

the packaging of functions and objects, their interfaces, and control to implement applications in a domain

whole-part diagram

a representation of aggregation, reflecting the whole and its component parts

References

[1] Adelson, B. and Soloway, E. “The Role of Domain Experience in Software Design.” *IEEE Transactions on Software Engineering* SE-11(11) (November 1985): 1351-1360.

[2] Alford, Mack W. “A Requirements Engineering Methodology for Real-Time Processing Requirements.” *IEEE Transactions on Software Engineering* 3(1) (January 1977): 60-69.

[3] Arango, G. and Prieto-Diaz, R. *Domain Analysis: Concepts and Research Directions*. 1990.

[4] Bach, William W. “Is Ada Really an Object-Oriented Programming Language.” *Journal of Pascal, Ada & Modula-2* 8(2), (Mar/Apr, 1989).

[5] Basili, V. “Viewing Maintenance as Reuse-Oriented Software Development.” *IEEE Software* 7(1) (January 1990): 19-25.

[6] Campbell Jr., G., Faulk, S., and Weiss, D. *Introduction to Synthesis*. (INTRO_SYNTHESIS-90019-N) Herndon, Virginia: SPC, June, 1990.

[7] Coad, P., and Yourdon, E. *Object-Oriented Analysis*. Englewood Cliffs, New Jersey: Prentice-Hall, 1990.

[8] Coad, Peter. *New Advances in Object-Oriented Analysis*. (IV-400). Austin, Texas, 1990.

[9] CTA, Incorporated. *JIAWG Reuse System Description*. (CDRL B004 (GMAX)). Ridgecrest, California: CTA, Incorporated, September 1990.

[10] CTA, Incorporated. *JIAWG Prototype Library Test Results.* (CDRL B004 (GMBF)). Ridgecrest, California: CTA, Incorporated, September 1990.

[11] D’Ippolito, Richard S. “Using Models in Software Engineering.” *Proceedings of Tri-Ada ‘89* (October 1989): 256-265.

[12] Department of Defense. *Defense System Software Standard.* Washington, D.C.: Department of Defense, 1988.

[13] Gilroy, Kathleen A., Comer, Edward R., J. Kaye Grau, J. Kaye, and Merlet, Patrick J. *Impact of Domain Analysis on Reuse Methods*. Indialantic, Florida: Software Productivity Solutions, 1989.

[14] Gish, James W. and Prieto-Diaz, Ruben. *Domain Analysis: Procedural Model Refinement and Experimental Proposal*. (87-126.05). Waltham, Massachusetts: GTE Laboratories, April 1988.

[15] Harel, David. “On Visual Formalisms.” *Communications of the ACM* 31(5) (May 1988): 514-530.

[16] Hoare C. A. R., ed. *Internation Series in Computer Science: Object-oriented Software Construction*. Hemel Hempstead: Prentice Hall, 1988.

[17] Holibaugh, Robert R., Cohen, Sholom G., Kang, Kyo C., and Peterson, Spencer. “Reuse: Where to Begin and Why.” *Proceedings of Tri-Ada ‘8*9 (October 1989): 266-277.

[18] Lanergan, R.G. and Poynton, B.A. “Reusable Code: The Application Development Technique for the Future.” *Proceedings of the IBM SHARE/GUIDE Software Symposium*. (October, 1979).

[19] Lee, K. and Rissman, M. *An Object-Oriented Solution Example: A Flight Simulator Electrical System*. (CMU/SEI-89-TR-5). Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University, February 1989.

[20] Matsumoto, M. “Reusable Software Parts Paradigm and Automatic Program Synthesis Using Them.” *Proceedings of the National Conference on Software Reusability and Maintainability*. (September 1986).

[21] McNicholl, Daniel G., et. al. *Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results*. (AFATL-TR-85-93). St. Louis, Missouri: McDonnell Douglas Astronautics Company, May 1986.

[22] McNicholl, D., et. al. *Common Ada Missile packages*. (AFATL-TR-85-93). Eglin AFB, Florida: Air Force Armament Laboratory, May 1986.

[23] McNicholl, D., Palmer, C., and Cohen, S. *Common Ada Missile Program--Phase2*. (AFATL-TR-88-62). Eglin AFB, Florida: Air Force Armament Laboratory, November 1988.

[24] Mish, Fredick, ed. *Webster’s Ninth New Collegiate Dictionary*. Springfield, Massachusetts: Merriam-Webster, 1985.

[25] Moore, J. and Bailin, S. *Domain Analysis: A Framework for Reuse*. Rockville, Maryland: CTA, INCORPORATED, October 1989.

[26] Parnas, D. “Desiging Software for Ease of Extension and Contraction.” *IEEE Transaction on Software Engineering* SE-5(2) (March 1979):128-138.

[27] Plinta, C., Lee, K., and Rissman , M. *A Model Solution for C3I Message Translation and Validation*. (CMU/SEI-89-TR-12), Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University, December 1989.

[28] Prieto-Diaz, Ruben. “Domain Analysis for Reusability.” *Proceedings of COMPSAC 87:The Eleventh Annual International Computer Software & Applications Conference* (October 1987): 23-29.

[29] SPC. *A Domain Analysis Process*. (Domain_Analysis-90001-N). Herndon, Virginia: Software Productivity Consortium, January 1990.