

LeanMBASE Additional SSAD Guideline



Center
For
Software
Engineering

General permission to make fair use in teaching or research of all or part of these guidelines is granted to individual readers, provided that the copyright notice of the Center for Software Engineering at the University of Southern California is given, and that reference is made to this publication. To otherwise use substantial excerpts or the entire work requires specific permission, as does reprint or republication of this material.

Sections

SECTIONS II

VERSION HISTORY.....III

TABLE OF CONTENTS IV

TABLE OF TABLES..... VI

ADDITIONAL SSAD GUIDELINE 7

Version History

Date	Author	Version	Changes made
08/20/05	Barry Boehm, David Klappholz, Ed Colbert, Prajakta Puri	1.0	<ul style="list-style-type: none">Initial version
08/17/05	Barry Boehm, David Klappholz, Prajakta Puri	1.1	<ul style="list-style-type: none">Updated the document for compatibility with LeanMBASE guideline v1.1.
08/28/05	Barry Boehm, Alex Lam, Prajakta Puri	1.2	<ul style="list-style-type: none">Updated section 2.3.1, 3.1.4.1.4, 4.1.4.1.4Updated table 1Deleted section 3.1.4.1.5, 1.1.4.1.5, 4.1.7.1.8, 4.1.8.1.3
10/02/05	Ed Colbert, Prajakta Puri	1.3	<ul style="list-style-type: none">Section 2 updated
10/14/05	Ed Colbert, Prajakta Puri	1.4	<ul style="list-style-type: none">Sections 2, 3 and 4 updated

Table of Contents

SECTIONS	II
VERSION HISTORY.....	III
TABLE OF CONTENTS	IV
TABLE OF TABLES.....	VI
ADDITIONAL SSAD GUIDELINE	7
A. Document Sections 7	
1. Introduction	7
1.1 Purpose of the SSAD Document	7
1.2 Standards and Conventions.....	7
1.3 References	7
2. System Analysis	7
2.1 Structure.....	7
2.2 Artifacts & Information	9
2.3 Behavior.....	11
3. Platform/Technology–Independent Model	16
3.1 Structure.....	17
3.2 Information Classes	32
3.3 Behavior.....	35
3.4 Architectural Styles, Patterns & Frameworks	38
4. Platform/Technology–Specific Model.....	38
4.1 Structure.....	39
4.2 Behavior.....	56
4.3 Patterns & Frameworks	58
4.4 Project Artifacts	58

Additional SSAD Guideline

5. Glossary for System Analysis and Design.....	61
6. Appendices	61

Table of Tables

<i>Table 1: Use-Case Description</i>	12
<i>Table 2: Typical Course of Action</i>	12
<i>Table 3: Alternate Course of Action: Name</i>	12
<i>Table 4: Exceptional Course of Action: Name</i>	13
<i>Table 5: Available Capabilities & Processes in Mode</i>	14
<i>Table 6: Level of Service Specification Form</i>	23
<i>Table 7: Parameter Table for Classifiers</i>	27
<i>Table 8: Component Processes Available in Mode</i>	29
<i>Table 9: Map of Connector Constraints to Sources and Elements</i>	31
<i>Table 10: Use-Case Realization Description</i>	36
<i>Table 11: Architecture Styles, Patterns, and Frameworks</i>	38
<i>Table 12: Class Attributes</i>	52
<i>Table 13: Operation Description</i>	53
<i>Table 14: L.O.S. Values for a Classifier Instance</i>	56
<i>Table 15: Stereotypes for Project Artifacts</i>	60

Additional SSAD Guideline

A. Document Sections

1. Introduction

1.1 Purpose of the SSAD Document

No additional details.

1.2 Standards and Conventions

No additional details.

1.3 References

No additional details.

2. System Analysis

No additional details.

2.1 Structure

Representation:

UML's Static-Structure and Collaboration Diagrams. Represent the model as a package with the stereotype <<Structure Model>> and the name "System Context". Represent the system as a classifier with the stereotype <<system>>. Represent each worker and outside actor, with which the system interacts, as a classifier with the stereotype <<actor>> (called an "actor"). The label of each classifier includes the name of the system, the worker, or the outside actor that it represents. Connect the classifier representing the system and each actor with a non-directional association that has the stereotype <<communication>>.

If an actor is a specialization of another actor (e.g. `Student` and `Library User`), then show a generalization relation from the specialized actor to the general actor. (In which case, it is unnecessary to show the association between the system and the specialized actor. The specialized actor inherits the association from the generalized actor.)

If there are a large number of the actors, then it is often useful to arrange the actors into packages that represent the sub-organization to which the actors belong. For example, a `Customer Service Representative` typically belongs to a `Customer Service` group within the organization. Each package should have the stereotype <<organization unit>>.

The UML Collaboration Diagrams show the particular configurations of the system and actor instances, possibly for a specific purpose. Each diagram shows instances of the system and the actor that participates

Additional SSAD Guideline

in that configuration using the classifier–instance notation (i.e. classifier symbol with a label of the form “instance name : classifier name” or “: classifier name”, and with the stereotype of the named classifier). If two instances interact in this configuration, then show a *link* connecting the two instances.

Each Static–Structure Diagram and Collaboration Diagram should be accompanied by a brief description of its purpose.

(For many systems, a single Static–Structure Diagram and a single Collaboration diagram are sufficient.)

For the system and each actor identified, create a subsection numbered 2.1.X (see example section below) and the name of the worker or outside actor in the header. The text of the subsection should give a brief description of worker or outside actor.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Give the system classifier and each actor a name that expresses the responsibilities of its instances.
 - A good name is usually a noun (e.g. “librarian”), a short noun phrase (e.g. “department secretary”), or the noun form of a verb (e.g. “administrator”).
 - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
2. Most system and actor instances will be unnamed. The instance’s label will be of the form “: classifier name”, which can be read as “some”, “a”, or “any instance of the named classifier”. For example, an instance with the label “: customer service representative” should be read as “any *customer service representative*”.
3. To facilitate traceability, assign each actor a unique number (e.g. Actor-10).

Common Pitfalls:

- Including artifacts (SSAD 2.2) in the context description.
- Including organization workers or outside actors that do not interact with the system according to the system’s behavior description (SSAD 2.3).
- Including components of the system. (They should be documented in SSAD 3.1).

Model Integration Rules:

- Each actor should be needed to implement a system capability or interface requirement (SSRD 3 and 4).
- LCA: each actor shown here shall be used to describe the system’s behavior (SSAD 2.3).

Additional SSAD Guideline

577 Guidelines:

Use UML's Static-Structure and Collaboration Diagrams, as described above, to define the system context.

2.1.1 System

Model Integration Rules:

- Each service shown in OCD 2.5 should be listed here.
- Each service should be represented as a system capability (OCD 3.1).
- The use of each service should be described in the system's behavior (SSAD 2.3).

2.1.2 Actor X

Representation

Refer section 2.1

2.2 Artifacts & Information

Representation

The *Artifact-Information* Model using the UML's Static-Structure Diagrams, which shows the classes of artifacts and information, and the potential relations among the classes. Each class artifact and information is represented as a class. If any one artifact and information class is a specialization of another class (e.g. `Tax Report` and `Report`), then show a generalization relation from the class representing the specialized artifact and information to the class representing the generalized artifact and information.

If two artifact and information classes are related other than by specialization, then an association is drawn to connect the two. The association should be labeled with the name of the relation. Each end of the association may be labeled with the role that artifact plays in the relation and the number of instances of the artifact that can be related to one incidents of the artifact at the other end of the association.

If there are a large number of artifacts or if some artifacts are only used by certain groups within the organization, then it is often useful to arrange the artifact classifiers into packages that represent the organizational units. For example, a `Customer Record` typically belongs to a `Customer Service` group within the organization.

Represent the model as a package and the name "System Information Model" with one or more Static-Structure Diagrams. Each Static-Structure Diagram should be accompanied by a brief description of its purpose.

For each artifact and information class represented, create a subsection numbered 2.2.x (see example section below) and the name of the artifact and information in the header. The text of the subsection should give a brief description of artifact and information class.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Additional SSAD Guideline

Recommendations:

1. Give each class a name that expresses the artifact and information that it represents.
 - A good name is usually a noun or short noun phrase. Avoid phrases that imply state of an object, e.g. “completed document”.
 - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
2. To facilitate traceability, assign each artifact and information class a unique number (e.g. Artifact-01 or SA-01)
3. Focus on top-level artifacts, e.g. `Sales Item` and a `Catalog of Sales Items` for a `Store`. Additional artifacts and their details can be provided in Technology-Independent Architecture Model or Technology-Specific Architecture Model (SSAD 3 and 4).
4. Focus on top-level information kinds, e.g. `Customer Record` and an `Employee Record` for a `Business`. Provided additional information kinds and their details (e.g. what is an `Address`) in Technology-Independent Architecture Model or Technology-Specific Architecture Model (SSAD 3 and 4 respectively).
5. Information about actors (SSAD 2.1) that the system needs to maintain (e.g. `user account` with attributes `user id` and `password`).

Common Pitfalls:

- Including structural elements (SSAD 2.1) in the Artifact-Information Model.
- Including an artifact and information kind that the system does not inspect, manipulate, produce, or maintain according to the system’s behavior (SSAD 2.3) at LCA.
- Including an artifact and information class that is not represented (called *realized*) in both in Technology-Independent Architecture Model or Technology-Specific Architecture Model (SSAD 3 and 4 respectively) at IOC.
- Including system *components*, which are structural elements that should be documented in the system in Technology-Independent Architecture Model (SSAD 3)
- Including design details about the artifacts and information class. (They should be deferred to in Technology-Independent Architecture Model or Technology-Specific Architecture Model (SSAD 3 and 4 respectively).)

Model Integration Rules:

- Each artifact and information kind should be needed to implement a system capability or interface requirement (SSRD 3 and 4 respectively).
- LCA: At least one instance of each artifact and information class should be used to describe the system’s behavior (2.3).

Additional SSAD Guideline

For each system's process (2.3) that is implemented at LCA, each artifact and information class used to describe that process (2.3) shall be realized in both in Technology-Independent Architecture Model or Technology-Specific Architecture Model (SSAD 3 and 4).

- IOC: Each artifact and information class shall be realized in Technology-Independent Architecture Model or Technology-Specific Architecture Model (SSAD 3 and 4).

2.2.1 Artifact or Information Class X

Representation

Refer section 2.2

2.3 Behavior

No additional details

2.3.1 Processes

Common Pitfalls:

- Simply creating one process for each capability (many capabilities will require more than one process since the capabilities were described a high-level, e.g. Maintain Vendor Profile).
- Confusing an Alternate or Exceptional Course of Action with a separate process.
- Describing a pre-condition then testing for that condition early in the Course of Action.
- Describing a post-condition than testing for that condition at the end of the Course of Action.
- Describing a condition has a precondition or post-condition when the condition must be tested in the Course of Action.

Model Integration Rules:

- Each system service described in SSAD 2.1.1 must be represented as a capability. (You may have more capabilities than services; but a capability that is not listed as service must specialize capability that is listed.)
- Each system capability described in OCD 3.1 must be represented as a capability here and must have at least one process that realizes it.
- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1.1).
- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information of the system (SSAD 2.2).

Additional SSAD Guideline

Table 1: Use–Case Description

Identifier	<i>Unique identifier for traceability (e.g. UC-xx)</i>
Use–Case Name	<i>Name of use–case</i>
Abstract	<i>Yes No</i>
Purpose	<i>Brief description of purpose</i>
Actors	<i>List of actors participating in the use–case</i>
Priority	<i>The relative importance of the process; using the MSCW (MoSCoW) prioritization scheme: M (Must have), S (Should have), C (Could have), W (Want to have)</i>
Capability:	<i>List of capabilities realized</i>
Requirements	<i>List of requirements that this use–case satisfies</i>
Risks	<i>List of risks for this use–case</i>
High–Risk?	<i>Yes No</i>
Architecturally Significant?	<i>Yes No</i>
Development Status	<i>Draft LCO Draft LCA Draft IOC LCO LCA IOC Accepted Build #</i>
Overview	<i>Overview of the behavior</i>
User Interface	<i>Pictures and/or descriptions of user interface (e.g. reference or URL to prototype screen), if applicable, that is needed to describe the process.</i>
Pre–conditions	<i>Description of the state that system and each participant should be in before use–case performed. (informal text, OCL, or both)</i>
Post–conditions	<i>Description of the system’s and participants’ states after use–case performed. (informal text, OCL, or both)</i>
Specializes	<i>List of use–cases that this use–case specializes</i>
Includes	<i>List of use–cases that are directly included by this use–case</i>
Extends	<i>Name of use–case extended by this use–case</i>
Extension Points	<i>List of names of extension points</i>

Table 2: Typical Course of Action

Seq. #	Actor Actions	System Response
1.	<i>Description of actor action</i>	
2		<i>Description of system response</i>
3	<i>Description of actor action</i>	
4		<i>Description user response</i>
	<i>etc.</i>	

Table 3: Alternate Course of Action: Name

Seq. #	Actor Actions	System Response
1.	<i>Description of actor action</i>	
2		<i>Description of system response</i>
3	<i>Description of actor action</i>	
4		<i>Description user response</i>
	<i>etc.</i>	

Additional SSAD Guideline

Table 4: Exceptional Course of Action: Name

Seq. #	Actor Actions	System Response
1.	<i>Description of actor action</i>	
2		<i>Description of system response</i>
3	<i>Description of actor action</i>	
4		<i>Description user response</i>
	<i>etc.</i>	

Recommendations:

1. Create a use-case for each process needed to implement a capability. For example, the capability Maintain Vendor Profile may require processes: Add Vendor, Delete Vendor, and Change Vendor.
2. Use the “just do it” approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Start with the system’s actions used in the organization’s processes that are associated with each capability (see the “Used In” field of the capabilities description). Each process may require several iterations to get right. “Go with what you know”, then revise the set of processes and adjust descriptions, as needed.
3. Give each process a name that expresses the behavior that it represents.
 - A good name is usually a verb or verb phrase.
 - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
4. To facilitate traceability, assign each process a unique number (e.g. Process-01 or UC-01).
5. Each process should be described in enough detail to be testable.
6. For LCO, the following fields of Use-Case Description should be filled out for all use-cases: Name, Purpose, Actors, Importance, Requirements, Development Status, Pre-conditions, Post-conditions, and Includes.
7. For LCA, Typical, Alternate, and Exceptional Courses of Actions should be described for high-risk, architecturally significant, or particularly complex use-case. Fill-in the Includes & Extension Points fields if the courses of actions specify the inclusion of other use-cases or extension point, respectively.
8. For IOC, use-cases descriptions should be completely filled out for all use-cases.

2.3.1.1 Process X

No additional details

2.3.2 Modes of Operation

Some systems respond differently to the same stimulus depending on the operational mode. For example, a voice-operated computer system may have two operational modes:

- *Operational Mode*: where the operators use the system to perform productive work.
- *Training Mode*: where the operators train the system to interpret their pronunciation of the commands to be used in the *Operational Mode*.

In *Operational Mode*, the voice-operated computer system would respond to the voice stimulus “Quit Application” would be to do so. In *Training Mode*, the system response to the same stimulus might be to ask the operator what action should be taken when “Quit Application” is spoken while in *Operational Mode*.

The term *mode* is often used for the high-level states of a system, which control what the system does in a broad way. (Low-level states typically control small details of the operation.) The mode defines a set of states that the system can be in and another that it cannot. For example, an airplane in *taxi* state (“mode”) can not be in the state *landing gear retracted*; but can be in the *landing gear brakes applied* and *airbrakes applied* states.

For simple systems, the system can only be in at most one state (mode) at a time. More complex systems may have nested and concurrent sub-states (sub-modes). For example, an airplane in the *taxi* state (“mode”) may be in the *landing gear brakes applied* and *airbrakes applied* substates simultaneously.

Representation:

Represent each mode as a state. If an event causes a mode change, then draw a transition line from the mode system is in when the event occurs to the mode to system will enter. Label the transition with the name of the event that must occur and any condition that must be met for the mode change to occur. Indicate the state in which the system begins (*initial state*) by placing the *initial pseudostate* on the diagram, and by drawing a transition from the initial pseudostate to initial state. If there is a *final state*, show it as an unlabeled “bull’s eye” symbol, and add appropriate transitions from other states to the final state.

For each mode, create a subsection numbered 2.3.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list system capabilities (OCD 3.1 and processes (SSAD 2.3.1) are available in that mode (see Table 5). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode-specific behavior).

Table 5: Available Capabilities & Processes in Mode

Capability	Processes	Mode Impact
<i>Capability id & name.</i>	<i>Process id & name</i>	<i>Describe any mode-specific behavior or L.O.S. goals</i>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Additional SSAD Guideline

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the system classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Modelx” that is attached to the system classifier, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

Recommendations:

1. Each state should have clearly defined entry and exit events. State models should be deterministic, i.e. if you are in state A and an event occurs, then it should be clear what state will become the current state. State models can be non-deterministic; but it is not recommended.
2. Keep the state model simple (i.e. only high-level modes) unless your system has high safety or reliability requirements.
3. Give each state a name that expresses the situation that it represents.
 - A good name is the present participle form of a verb (e.g. “waiting”, “taxing”, “flying”) or the past participle form of a verb (e.g. “stopped”, “completed”), or an adjective (e.g. “active”, “passive”, “draft”, “final”).
 - Each name must be unique relative to the containing state model. (Two states in different models or substates in different states.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
4. Give each event a name that expresses the “thing that happened”. Events include asynchronous stimuli between two instances (*signals*), operation invocations (*calls*), the passage of time, or a change in state. Events may be synchronous or asynchronous.
 - A good name for a signal or for call is the name of the operation being requested.
 - Each name must be unique relative to the containing system model.
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
 - A time event should be of the form “**after** (*x units*)”, “**when** (date = *xxxxx*)”, or “**when** (time = *xxxxx*)”.
 - A change of state should be of the form “**when** (*boolean expression*)”.

Common Pitfalls:

- Not specifying the initial state for a model.
- Confusing a state with an activity.
- Confusing an event with a condition.

Additional SSAD Guideline

Model Integration Rules:

- Each required mode (SSRD 6) should be represented as a mode here.
- Each requirement associated with are required mode (SSRD 6) should be realized by one or more of the processes associated with that mode.
- Each system capability (OCD 3.1) should be listed as available in one or more modes.
- Each system process (SSAD 2.3.1) should be listed as available in one or more modes.
- Each capability listed as available in one or more modes should appear in described in OCD 3.1.
- Each process listed as available in one or more modes should appear in described in SSAD 2.3.1.

2.3.2.1 *Mode X*

List in a table the system capabilities (OCD 3.1) the processes (SSAD 2.3.1) associated with each capability that are available in this mode; describe any modal effects on the processes. For example,

Capability	Processes	Modal Impact
Capability-01 ...	Process-01 ...	
	Process-02 ...	LOS-01 only applies to this process in this mode.
Capability-03 ...	Process-05 ...	

3. Platform/Technology–Independent Model

The three categories of systems and their typical components:

- *System of systems* is a system that is a collection of loosely coupled systems, i.e. provide distinctly useful capabilities that can be used in various combinations or potentially alone. Each component is itself a system, whose architecture needs to be described unless reusing an existing system or purchasing a COTS or custom system. (An example of a System of Systems is a system that combines a navigation system, a vehicle–control system, a weapon–control system, and a life–support system.)
- *Composite system* is a system of separately executing components that are generally useful only when working with other components. At the architecture design level, the components are abstractions of the concurrent processes (also called “OS processes” or “tasks”), threads, databases, dynamic– or static–link libraries, CORBA or COM objects, or Enterprise Java Beans, which will be used in the implementation.
- *Simple system* is typically a system that is implemented as a single software executable running on a single computer or device. Sample software components include the software executable and its *active* objects and classes if using an object–oriented development; functions if using a function–oriented development, and rules if using a rule–oriented development.

Recommendations:

The subsections of this section described below are intended to be general purpose. Tailor this section based on the size of the system, the chosen representation language, and the architectural style(s).

Additional SSAD Guideline

Model Integration Rules:

- The rationale for the Architecture must be documented in the FRD.

3.1 Structure

Notes:

Throughout Platform/Technology–Independent Model (SSAD 3) and Platform/Technology–Specific Model (SSAD 4), the term *component classifier* is used for the kind of component (e.g. human), and the term *component* is used for the instance of component classifier (e.g. you).

3.1.1 Hardware Classifier Model

Representation:

UML's Deployment Diagrams. Create one or more UML Deployment Diagrams that shows the kinds of hardware as node classifiers with the name of the hardware type. If instances of two kinds of hardware can be connected (e.g. wire together), then connect their node classifiers with an association that has the stereotype <<connector>> and whose name is the class of the connector (e.g. "Bus", "Network").

Each Deployment Diagram should be accompanied by a brief description of its purpose.

(For many systems, a single Deployment Diagram is sufficient to represent the hardware classes.)

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Give the hardware component classifier a name that expresses the kind of hardware.
 - A good name is usually a noun (e.g. "Workstation"), a short noun phrase (e.g. "Static RAM"), or the noun form of a verb (e.g. "Server").
 - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words (e.g. "UNIX Server" "Pentium IV").
2. To facilitate traceability, assign each component a unique number (e.g. Component-10, HWCC-06).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Including software component classifiers. (They should be documented in SSAD 3.1.6)
- Including actors that are not described in the system context (SSAD 2.1).

Additional SSAD Guideline

Model Integration Rules:

- Each actor described here should be described the system context (SSAD 2.1).
- LCO: Describe classifiers of expected hardware components.
- LCA: complete descriptions of all hardware component classifier needed to support high-risk or architecturally-significant behaviors.
- IOC: at least one instance of each hardware component classifier shown here shall appear in the Deployment Model (SSAD 3.1.3).

577 Guidelines:

Rose does support the current UML semantics for the Deployment Diagram. Create one or more UML Static-Structure Diagrams that shows the node types represented as classifiers with the stereotype <<node>> and the name of the hardware kind, and shows the actors that interact with the hardware components. If instances of two hardware classifiers, or actor and a hardware classifier, can be connected (e.g. wire together), then connect their node classifiers with an association that has the stereotype <<connector>> and whose name is the class of the connector (e.g. “Bus”, “Network”).

3.1.2 Software Classifier Model

Representation:

UML’s Component Diagrams. Create one or more UML Component Diagrams that shows the kinds of software components, their interfaces, their relations, and the classes and objects that reside on the component. Each kind of software components is represented as component classifiers with the name of the kinds of software components. Interfaces are represented as a circle (the stereotypical icon form) with an association connecting it to component that realizes it.

UML v1.4 [OMG 2001] does not define the concept of “connectors” for software components. If the instances of component classifier requires services from instances of another component classifier,

If the interfaces have been defined (see Interface(s) (SSAD 3.1.6.1.2)) for the component classifier, then a dependency relation should be drawn from the component classifier that needs the service to the appropriate interface(s) of the supplying component classifier.

Otherwise, a dependency relation should be drawn from the component that needs the service to the supplying component.

Each Component Diagram should be accompanied by a brief description of its purpose.

(For many systems, a single Component Diagram is sufficient to represent the software classes.)

For each software component classifier, create a subsection numbered 3.1.6.x with the name of the hardware connector classifier in the header, under the Software Component Classifiers (SSAD 3.1.6). The body of the subsection should describe pertinent information about the kind of software.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Additional SSAD Guideline

Recommendations:

1. Give each software component classifier a name that expresses the kind of software.
 - A good name is usually a noun (e.g. “Administrator”), a short noun phrase (e.g. “User Agent”), or the noun form of a verb (e.g. “Controller”), that is either a:

Common noun or noun phrase, i.e. the name of a class of persons, places, or things (e.g. “Apple”, “Building”, “F16”, “Computer”),

Mass or abstract noun or noun phrase, i.e. the name of a quality, activity, or substance (e.g. “water”, “traffic”, “software”, “cooking”, “usefulness”, “programming”, “completeness”),

Countable noun or noun phrase, i.e. the name of a thing that can be enumerated (e.g. “targets”, “code”)

Either mass nouns or countable nouns may be modified by a unit of measure, i.e. means of referring to a quantity (e.g. “water volume in cubic centimeters”, “percent complete”, “number of targets”, “lines of code”).

- Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words (e.g. “Library User”, “System Administrator”).
2. To facilitate traceability, assign each component a unique number (e.g. Component-10, SWCC-05).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Including actors that are not described in the system context (SSAD 2.1).
- Including Technology-Specific Architecture component classifiers or details, e.g. specific kinds of computer at LCO. (They should be documented in SSAD 4.1)

Model Integration Rules:

- Each actor described here should be described in the system context (SSAD 2.1).
- LCO: Describe classifiers of expected software components.
- LCA: complete descriptions of all software component classifiers needed to support high-risk or architecturally-significant behaviors.
- IOC: at least one instance of each software component classifier shown here shall appear in the Deployment Model (SSAD 3.1.3).

577 Guidelines:

Rose does not support the current UML semantics for the Component Diagram. Create one or more UML Static-Structure Diagrams that show the node types represented as classifiers with the stereotype

Additional SSAD Guideline

<<component>> and the name of the software kind. Interfaces are represented as a circle (the stereotypical icon form) with an association connecting it to component that realizes it.

If a component requires services from another component, whose interfaces have been defined (see Interface(s) (SSAD 3.1.6.1.2)), then a dependency relation should be drawn from the component that needs the service to the appropriate interface(s) of the supplying component. If a component requires services from another component whose interfaces for the supplying component have not been defined, then a dependency relation should be drawn from the component that needs the service to the supplying component.

(While UML v1.4 [OMG 2001] does not define the concept of “connectors” for software components, as described above, if you want to use the concept, you can connect two <<component>> classifiers with an association that has the stereotype <<connector>> and whose name is the class of the connectors (e.g. “adaptor”, “distributor”). The interface can be represented as the classifier for the role.

3.1.3 Deployment Model

Representation:

UML’s Deployment Model. Create a package with the stereotype <<Deployment Model>> with the name “Configuration Name”, where “Configuration Name” is the name of the configuration (e.g. “Standard Configuration”, “High Assurance Configuration”, “In-flight Mode”) described by the diagrams in the package.

Create one or more UML Deployment Diagrams that shows hardware components, the software components that reside on each hardware component, the connectors between hardware components, and dependency relations between software components.

Each hardware components is represented as a node classifiers with a label of the form “*instance_name* : *node_classifier_name*”, “*instance_name* / *role_nme* : *node_classifier_name*”, “/ *role_name* : *node_classifier_name*”, or “: *node_classifier_name*” (e.g. “Local / User Station : Workstation”, “/Administrator: Workstation”, or “ISD Server : UNIX Server”).

Each software components is represented as a component classifiers with a label of the form “*instance_name* : *node_classifier_name*”, “*instance_name* / *role_nme* : *node_classifier_name*”, “/ *role_name* : *node_classifier_name*”, or “: *node_classifier_name*” (e.g. “: Library_User”). The software components are drawn inside the hardware nodes.

If two hardware instances are connected (e.g. wire together), then connect their node classifiers with a link that has the stereotype <<connector>> with a label of the form “*instance_name* : *node_classifier_name*” or “: *node_classifier_name*” (e.g. “LAN : Network”).

If a software component requires services from another software component,

If the interfaces have been defined (see Interface(s) (SSAD 3.1.6.1.2)) for the supplying component, then a dependency relation should be drawn from the component that needs the service to the appropriate interface(s) of the supplying component.

Otherwise, a dependency relation should be drawn from the component that needs the service to the supplying component.

Each Deployment Diagram should be accompanied by a brief description of its purpose.

Additional SSAD Guideline

For each hardware component, create a subsection numbered 3.1.7.x with the name of the hardware component in the header, under the Hardware Components (SSAD 3.1.7). The body of the subsection should describe the hardware component.

For each software component, create a subsection numbered 3.1.8.x with the name of the software component in the header, under the Software Components (SSAD 3.1.8). The body of the subsection should describe the software component.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. If the software component has a name, the name should describe an instance of hardware or software.

- A good name is usually a noun, a short noun phrase, or the noun form of a verb, that is either a:

Proper noun or noun phrase, i.e. the name of a specific person, place, or thing (e.g. “UNIX”, “Internet”),

Direct reference, i.e. a reference to previously identified or known person, place, or thing without necessarily using name (e.g. “the user agent”, “the local workstation”, “the system administrator”).

(Proper nouns are uncommon at this stage of development.)

2. If the software component has a role name, the name should describe the part played by the instance in the configuration (e.g. “the user workstation”, “the database server”).

- Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
- Avoid names that sound alike or are spelled alike, as well as synonyms.
- Clear, self-explanatory names may require several words (e.g. “Library User”, “System Administrator”).

3. To facilitate traceability, assign each component a unique number (e.g. Component-10, SWCI-05).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Including implementation component classifiers or details, e.g. specific kinds of computer at LCO. (They should be documented in SSAD 4.1)

Model Integration Rules:

- Each software component classifier described here should be shown in the Software Classifier Model (SSAD 3.1.2) and described in the section Software Component Classifiers (SSAD 3.1.6).
- Each actor described here should be described in the system context (SSAD 2.1).

Additional SSAD Guideline

- At LCO, a System Deployment Model should show the expected configuration of hardware and software components, whose classifiers are described in the Hardware Classifier Model (SSAD 3.1.1) and the Software Classifier Model (SSAD 3.1.2).
- At LCA, the System Deployment Model shall show the final configuration of hardware and software that participate in high-risk behaviors, and the preliminary allocation of all other components.
- At IOC, the System Deployment Model shall show the final configuration of all components.

577 Guidelines:

Rose does support the current UML semantics for the Deployment Diagram. Create an UML *Object Diagram* (a UML Static-Structure Diagram with instances and no classifiers) within the “Deployment Model” package that shows the hardware instances and the hardware connection instances. Each node is represented as classifiers with the stereotype <<node>> and a label of the form “*instance_name* : *node_classifier_name*”, “*instance_name* / *role_name* : *node_classifier_name*”, “/ *role_name* : *node_classifier_name*”, or “: *node_classifier_name*”.

For each node instance that has a unique configuration of hardware or software residing on it, create a sub-Package of the “Deployment Model” package with the stereotype <<node>> and a label of the form “*instance_name* : *node_classifier_name*”, “*instance_name* / *role_name* : *node_classifier_name*”, “/ *role_name* : *node_classifier_name*”, or “: *node_classifier_name*”.

In each node package, create an UML Object Diagram that shows the component instances that reside on that node and the component instances that the resident components use. Each component instance is represented as a classifier with the stereotype <<component>> and a label of the form “*instance_name* : *node_classifier_name*”, “*instance_name* / *role_name* : *node_classifier_name*”, “/ *role_name* : *node_classifier_name*”, or “: *node_classifier_name*”. If the instance resides on this node, then the *instance_name* should be the simple name of the component; otherwise, the qualified name of the component. For each *component*, show the following.

- The interfaces of the component that that have been identified. (At LCA all component interfaces should be identified.)
- For each interface of the component, a realize relation from the component to its interface.
- For each component used, a uses relations to the interfaces of the used component if its interfaces have been identified, or to the used component if its interfaces have not been identified.

3.1.4 Hardware Component Classifiers

No additional details

3.1.4.1 Component Classifier X

No additional details

3.1.4.1.1 Purpose

No additional details

Additional SSAD Guideline

3.1.4.1.2 L.O.S. Characteristics Goals

Representation:

For each system goal that applies to this component class fill out the L.O.S. Form (Table 6). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its own value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

Table 6: Level of Service Specification Form

Level of Service:	<<Give a reference number and name>> such as “LS-1.1: Response time”
Description:	<<Describe the level of service>>, such as “1 second desired; 2 seconds acceptable”
Measurable:	<<Indicate how this goal can be measured with respect the specific elements it addresses – include as appropriate baseline measurements, minimum values, maximum values, average or typical or expected values, etc. >>, such as “time between hitting Enter and getting useful information on the screen”
Relevant:	<<Describe system process (SSAD 2.3.1), mode (2.3.2), or artifact or information classifier (2.2) to which this level of service is relevant>>, such as “larger delays in order processing (Process–07 in SSAD 2.3.1) cause user frustration”
Specific:	<<Describe what in particular within the system process (SSAD 2.3.1), mode (2.3.2), or artifact or information classifier (2.2) this level of service addresses>>, such as “credit card validation during order processing (Process–07 in SSAD 2.3.1) may cause significant delay when attempting to connect to the verification service”

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. Consider the system processes in which instances of this component classifier participates, when determining which system L.O.S. goals apply to this component classifier. L.O.S. goals that apply to the system processes (SSAD 2.3) generally apply to the components that participate in the process.
2. Consider derived goals, i.e. a system–level throughput requirement may mean imply a maximum time to complete one of this component classifier’s processes.
3. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

Common Pitfalls:

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.

Model Integration Rules:

- Each system L.O.S. Goals (OCD 3.3.3) should be described here.

Additional SSAD Guideline

- Each artifact or information classifier listed here, should be described in the architecture's Data Model (SSAD 3.2).

3.1.5 Hardware Connector Classifiers

No additional details

3.1.5.1 Connector Classifier X

No additional details

3.1.5.1.1 Purpose

No additional details

3.1.5.1.2 L.O.S. Goals

Representation:

For each system goal that applies to this connector fill out the L.O.S. Form (Table 6). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its own value, then specify that the "value is specific to the component instance". Optionally specify a default value or a limiting value.

Recommendations:

1. Consider the system processes in which the components connected by this connector participates when determine which system L.O.S. goals apply to this connector. L.O.S. goals that apply to the system processes (SSAD 2.3) generally apply to the connectors that participate in the process.
2. Consider derived goals, i.e. a system-level throughput requirement may mean imply a maximum time to complete one of this connector's processes (e.g. transporting a message).
3. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

Common Pitfalls:

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.

Model Integration Rules:

- Each connector L.O.S. Goals (OCD 3.3.3) should be described here.
- Each artifact or information classifier listed here, should be described in the architecture's Data Model (SSAD 3.2).

3.1.6 Software Component Classifiers

Model Integration Rules:

- At LCO: Draft components are identified. Each component shall have a clear statement of purpose.
- At LCA: Components and interfaces of components that are high-risk shall be described in sufficient detail to support evaluation of correctness, and have a draft implementation design (SSAD 4.1).
- At IOC: All components and interfaces of components in the build shall be described in sufficient detail to support evaluation of correctness, and have a designed implementation (SSAD 4.1).

3.1.6.1 Component Classifier X

No additional details.

3.1.6.1.1 Purpose

No additional details.

3.1.6.1.2 Interface(s)

Representation:

UML's Static-Structure Diagrams & Component Diagrams. A set of services is called an interface, which is represented as a classifier with the stereotype <<interface>> and the name of the set (e.g. "Security Services" or "File Management"). Note: an interface in UML is not allowed to have either attributes or parts.

Create a UML Static-Structure Diagram named "Interface Details" that defines the interfaces of the software component classifier, the operations they contained, and any inheritance relations among them. (Note: these interface classifiers may be used to define multiple components, connectors, and classes.)

Create a UML Component Diagram named "Interfaces" that shows the component classifiers for the component being described and its visible subcomponents; the interface classifiers (they will appear in icon form) define in the Static-Structure Diagram, and its visible attributes. Connect the component representing the component being described to each interface with an association. To represent a visible subcomponent to the component being described, either (a) connect the component representing the subcomponent being described using a composition relation to each component that represents a subcomponent; or (b) graphically nest each component classifier representing a subcomponent in the component classifier representing the component being described. Set the visibility of each subcomponent to public. (Operations in interfaces are always public.)

(UML does not allow attributes for the components.)

Note: the component's Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Component Diagram. (Name the diagram "Interfaces & Parameters".)

Additional SSAD Guideline

For each feature or set of features, create a subsection numbered 3.1.6.1.2.x and the name of the software feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Model Integration Rules:

- Each interface feature described here, should be used in the component's behavior description (SSAD 3.1.6.1.4).
- At LCA: a preliminary set of interfaces for the component shall be described.
- At IOC: the final set of interfaces for the component shall be described.

577 Guidelines:

Rose does support the current UML semantics for the Component Diagram, including some of what is described above. Create the Static-Structure Diagram named "Interfaces".

- Add a classifier with the stereotype <<component>> for the component being described and for each of its visible subcomponents.
- Add a classifier with the stereotype <<interface>> and the name of the interface for each set of related operations. Define the operations they contained and show any inheritance relations among the interfaces.
- Connect the <<component>> classifier representing the component being described to each interface with a realization relation.
- Add any attributes of the software component to the <<component>> classifier.
- Add a classifier with the stereotype <<component>> for each of its visible subcomponents.
- Add a composition relation from the <<component>> classifier representing the component being described to each <<component>> classifier that represents a subcomponent. (If the <<component>> classifier representing the component being described does not have any attributes, then graphically nest each <<component>> classifier that represents a subcomponent inside the component representing the component being described can be an alternate representation for composition; but many tools do not recognize the relation.)
- Set the visibility of each subcomponent and each attribute to public. (Operations in interfaces are always public.)

Note: the component's Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Static-Structure Diagram. (Name the diagram "Interfaces & Parameters".)

3.1.6.1.2.1 *Feature or Feature Set X*

Refer section 3.1.6.1.2

3.1.6.1.3 Parameters

Representation:

UML's Component Diagrams. Create a UML Component Diagram that shows the component classifier for the component being described. Add a dashed rectangular box to upper-right hand corner of the component classifier. List the parameters in the box.

Additional SSAD Guideline

Note: the component's Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Component Diagram.

Create a table like Table 7 that lists each parameter.

Table 7: Parameter Table for Classifiers

Parameter Name	Type/Signature	Default Value	Purpose
<i>Name & Identifier</i>	<i>(see below)</i>	<i>What to use if no argument is bound to this parameter</i>	<i>A brief description of the purpose of the parameter.</i>

The Type/Signature field depends on the kind of parameter being described. The following paragraphs describe the some common contents for the Type/Signature field.

- If the parameter is an instance (e.g. an object, a component), then the contents must be the name of a classifier. If a value is expected, put "in" before the classifier name
- If the parameter is an operation, then the content is the operations signature.
- If the parameter is a classifier, then the contents are either blank (as in UML) or should say "Classifier".

(Note: the "kind" of the parameter is determined by the contents of the Type/Signature field.)

Model Integration Rules:

- At LCO: a preliminary set of parameters for the component, if any exist, shall be described.
- At LCA: updates to the set of parameters for the component identified at LCO, if any exist, shall be described.
- At IOC: the final set of parameters for the component shall be described. (If none, say so.)

577 Guidelines:

Rose does not fully support the current UML semantics for the Component Diagram, including what is described above. Create a Static-Structure Diagram instead of the UML Component Diagram described above. In the Static-Structure Diagram, create a classifier with the stereotype <<component>> for the component being described. Add a dashed rectangular box to upper-right hand corner of the classifier. List the parameters in the box.

Note: the component's Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Component Diagram.

3.1.6.1.4 Behavior

Model Integration Rules:

- At LCO: a draft behavior for the component shall be described if the component is high-risk.
- At LCA: updates to the behavior for the component identified at LCO, if any exist, shall be described.

Additional SSAD Guideline

- At IOC: the behavior for the component shall be described if the component is included in the build.

UML Guidelines:

Describe the processes of this component classifier, including how it works with the other components and the system's actors, and how it uses the artifacts and information; and if significant, how the component's behavior depends on its mode.

The following subsection work for UML-based architecture descriptions and may work for other ADL's.

3.1.6.1.4.1 Processes

Representation:

As in 2.3, create a Use-Case Model that describes the component's processes. Represent the model as a package with the stereotype <<use-case model>> with the package name "*Component Classifier Name Processes*".

Representation:

Create a Use-Case Model that describes the system's processes; the actors that participate in each process; and the relations among the processes and the outside actors. Represent the model as a package with the stereotype <<use-case model>> with the package name "System Processes".

Recommendations:

See Recommendations in 2.3.

1. Use the "just do it" approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Each process may require several iterations to get right. "Go with what you know", then revise the set of processes and adjust descriptions, as needed.
2. Start with the component's actions or requested operation described in the architecture behavior description (SSAD 3.3).
3. To facilitate traceability, assign each process a unique number (e.g. Process-01 or UC-01).
4. For LCA
 - The following fields of Use-Case Description should be filled out for all use-cases: Name, Purpose, Actors, Importance, Requirements, Development Status, Pre-conditions, Post-conditions, and Includes.
 - Typical, Alternate, and Exceptional Courses of Actions should be described for high-risk, architecturally significant, or particularly complex use-case. Fill-in the Includes & Extension Points fields if the courses of actions specify the inclusion of other use-cases or extension point, respectively.
5. For IOC, use-cases descriptions should be completely filled out for all use-cases.

Additional SSAD Guideline

Common Pitfalls:

See Common Pitfalls in 2.3.

Model Integration Rules:

- Each operation defined in the component's interface (SSAD 3.1.6.1.2) must either be represented by a use-case or represented as an action in a use-case's Activity Diagram.
- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).
- Each component in a process description should be defined in the architecture of the system (SSAD 3.1).
- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information for the system's architecture (SSAD 3.2).

3.1.6.1.4.1.1 Process X

Create a section at this level for each process of the component. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

3.1.6.1.4.2 Modes of Operation

Representation:

As in SSAD 2.3.2, create a State Model. Represent the model as a package with the stereotype <<state model>> with the package name "Modes of *Component Classifier Name*". Create one or more Statecharts that show the modes of component and the events that cause mode changes.

For each mode, create a subsection numbered 3.1.6.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list component's processes (SSAD 3.1.6.1.4.1) are available in that mode (see table 8). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode than in another, the process may have mode-specific behavior).

Table 8: Component Processes Available in Mode

Processes	Mode Impact
<i>Process id & name</i>	<i>Describe any mode-specific behavior or L.O.S. goals of the process</i>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the <<component>> classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named "State/Activity Modelx" that is attached to the <<component>> classifier, and a diagram named "NewDiagram" in the model. Rename the State Model to "Modes of Operation" and the diagram to "Top Level".

Additional SSAD Guideline

Recommendations:

See Recommendations in SSAD 2.3.2

Common Pitfalls:

See Common Pitfalls in SSAD 2.3.2

Model Integration Rules:

- Each component process (SSAD 3.1.6.1.4.1) should be listed as available in one or more modes.
- Each process listed as available in one or more modes should appear in described in SSAD 3.1.6.1.4.1

3.1.6.1.4.2.1 *Mode X*

No additional details.

3.1.6.1.5 Constraints

Representation:

Create a list of system rules described in either informal text or a formal specification language (e.g. UML's OCL). Clearly identify any system processes (SSAD 2.3), or artifact and information classifiers (SSAD 2.2) to which the rule is known to apply.

Create a table like that shown in table 9 that shows how organization rules are supported by system rules and to which system elements the organization and system rules. If an organization rule is not supported by a system rule, put "N/A" (not applicable) in the System Rule column.

Model Integration Rules:

- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each process described in a constraint, should be described in the component classifier's Processes (SSAD 3.1.6.1.4.1).
- Each mode described in a constraint, should be described in the component classifier's Modes of Operation (SSAD 3.1.6.1.4.1).
- Each artifact or information classifier described in a constraint, should be described in the architecture's Information Classes (SSAD 3.2).
- IOC: Each component rule should be implemented by one or more elements described in either this component's Internal Architecture (SSAD 3.1.6.1.7), or its Technology-Specific Model (SSAD 4).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Common Pitfalls:

- Describing a rule that is already captured in another view of the component

Additional SSAD Guideline

Table 9: Map of Connector Constraints to Sources and Elements

Trace's Back To	Connector Rule	Applies To
<i>Identifier of System Rule or Architecture Style, Pattern Or Framework</i>	<i>Connector Rule Identifier</i>	<i>List of connector's processes, modes, or artifact or information classifiers</i>

3.1.6.1.6 Internal Architecture

Representation:

Apply the same guidelines to describe the architecture of a component as was used to describe for the system architecture in section Technology-Independent Architecture Model (SSAD 3), i.e. create subsections: "Structure", "Information Classes", "Behavior", and "Architectural Styles, Patterns & Framework".

Recommendations:

1. Only specify the internal architecture for a software component when the component will to be implemented by the project or when the internal architecture is needed to analyze the system. For a Small System or Composite System, the internal architectures of software components are often not important.
2. You need not specify the architecture of a component which according to your architecture language or style is the lowest level that can be specified (i.e. "atomic"). In UML, an atomic component is the lowest-level modular, deployable, and replaceable part of system (e.g. the representation of an executable, a link-library, a Java Bean).

A complete architecture specification would define an internal architecture for all but the atomic components. However, schedule or cost constraints may stop the architecture design process before uniformly reaching all atomic components. A project may also stop if it determines that there is little benefit from specifying all atomic components.

Common Pitfalls:

- Spending time describing the internal architecture for low risk components, including components that will not be implemented by this project. (This problem is more common for Simple Systems and small Composite Systems.) If the component is low risk, consider deferring its details until Technology-Specific Architecture Model (SSAD 4).

Model Integration Rules:

- At LCA: an internal architecture for each non-atomic component shall be described if the component is high-risk or complex.
- At IOC: an internal architecture for each non-atomic component shall be described if the component is included in the build.

577 Guidelines:

While your system is probably a Composite System, it is probably simple enough, that you are unlikely to need this detailed of a description for your software.

Additional SSAD Guideline

If you decide the component's internal architecture needs to be specified, create a UML package with the stereotype <<Structure Model>> the name "Architecture" in the package representing this component (i.e. the one with stereotype <<component>> and the name "*Component Classifier Name AD*") to hold the internal architecture description.

3.1.7 Hardware Components

No additional details.

3.1.7.1 Component X

No additional details.

3.1.7.1.1 Purpose

No additional details.

3.1.7.1.2 Classifier

No additional details.

3.1.8 Software Components

No additional details.

3.1.8.1 Component X

No additional details.

3.1.8.1.1 Purpose

No additional details.

3.1.8.1.2 Classifier

No additional details.

3.2 Information Classes

Representation

UML's Class Model. Represent the model as a package with the stereotype <<Class Model>> and the name "Information Classes". In the model Information Classes packages, create one or more Static-Structure Diagrams to show the information classes that are needed to support the architectural structure implement the system behavior, and their relations.

Represent each artifact or information classifier that is stored, or used for internal communication as a classifier with the stereotype <<entity>>. Represent each form or other information that is used to

Additional SSAD Guideline

communicate with an actor as a classifier with the stereotype <<boundary>>. (Some artifacts and information classifier may have both <<entity>> and <<boundary>> representations.)

If one class is a specialization of class (e.g. `Tax Report` and `Report`), then show a generalization relation from the classifier representing the specialized class to the classifier representing the generalized class.

If two classes are directly related other than by specialization, then an association is drawn to connect the two classifiers. The association should be labeled with the name of the relation. Each end of the association may be labeled with the role that class plays in the relation and the number of instances of the class that can be related to one incidents of the class at the other end of the association.

If two classes are not directly related; but one needs the other (e.g. as the class of a parameter), then a dependency relation is drawn from dependant class (“client”) to other class (“supplier”). The dependency relation may have a name (uncommon) and a stereotype.

If there are a large number of classes, then it is often useful to organize the classes and diagrams using packages.

Each Static–Structure Diagram should be accompanied by a brief description of its purpose.

For each class represented, create a subsection numbered 3.2.X and the name of the artifact in the header. The text of the subsection should give a brief description of artifact or class of information.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Give each classifier a name that expresses the artifact or information that it represents.
 - A good name is usually a noun or short noun phrase. Avoid phrases that imply state of an object, e.g. “completed document”.
 - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
2. To facilitate traceability, assign each class a unique number (e.g. Entity-01 or Boundary-01)
3. Focus on classes and their details that are needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture. Provided additional class and details in Technology–Specific Architecture Modeling (SSAD 4).
4. Create additional Static–Structure Diagrams that show how each artifact and information classifier defined during system analysis (SSAD 2.2) is implemented by one or more analysis classes and how the information that the system needs to maintain about an actor is represented by one or more analysis. Represent each system artifact and information classifier as described in system analysis (SSAD 2.2), and each of the architecture’s analysis class as described under Representation in this section. For each analysis class that implements a system artifact or information classifier, show a dependency relation with the stereotype <<trace>> from the analysis class to the system artifact or

Additional SSAD Guideline

information classifier that the analysis class implements. For each analysis class that represents information about an actor, show a dependency relation with the stereotype <<trace>> from the analysis class to the actor whose information the analysis class represents.

5. Suppress the display of all (or all but the most important) attributes and operations on the Static-Structure Diagrams. Showing attributes or operations of the classes on the Static-Structure Diagram can result in diagrams that are too cluttered and so difficult to understand.

Common Pitfalls:

- Including components (SSAD 3.1).
- At LCA:

Not representing all artifact and information classifiers defined for the system (SSAD 2.2).

Including details about a class that are not needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture. (They should be deferred to Technology-Specific Architecture Model (SSAD 4).)

- At IOC:

Including classes that are not needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture.

Including classes that are not represented (called *realized*) in Technology-Specific Architecture Model (SSAD 4).

Model Integration Rules:

- LCO:

Each artifact and information kind artifact and information classifiers defined for the system (SSAD 2.2) should be represented.

Each class needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture should be represented.

- LCA:

Each artifact and information classifiers defined for the system (SSAD 2.2) shall be represented here.

Each class needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture shall be represented.

- IOC:

At least one instance of each class shall be used to describe either the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture.

Every detail about a class shall be used to describe either the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture.

Additional SSAD Guideline

Each class shall be realized in Technology-Specific Architecture Model (SSAD 4).

577 Guidelines:

Create an Information Class Model using a UML Class Model as described above.

3.2.1 Information Class X

Representation:

Refer Section 3.2

3.3 Behavior

Representation:

Create a Use-Case Model that describes the system's processes, the actors that participate in each process and the relations among the processes and the outside actors. Represent the model as a package with the stereotype <<use-case model>> with the package name "Process Implementations".

Create one or more *Use-Case Diagrams* that show the system processes use-case, the process implementations that implement them, and a *realization* relation from each process implementation to the process that it implements. Represent each process as a basic use-case, and each implementation as a use-case with the stereotype <<use-case realization>>. (Some processes may have multiple implementations. For example, a `Login` process may have "Basic" implementation and a "Secure" implementation)

Create a *Use-Case Realization Description*, and one or more *Interaction Diagram*.

For each high priority process create a Use Case Model, Use-Case Realization Description (Table 10) and one or more Interaction Diagram. For other process it's sufficient to just have Use-Case Description (Table 10)

Common Pitfalls:

- Simply creating one process for each capability (some processes will require more than one implementation).
- Describing a pre-condition then testing for that condition early in the Course of Action.
- Describing a post-condition than testing for that condition at the end of the Course of Action.
- Describing a condition has a precondition or post-condition when the condition must be tested in the Course of Action.

Table 10: Use-Case Realization Description

Identifier	<i>Unique identifier for traceability (e.g. UCR-xx)</i>
Use-case Realization Name	<i>Name of process implementation</i>
Use-case Realized	<i>Name & identifier of processes</i>
Purpose	<i>Brief description of purpose (include how it is different from other implementations)</i>
Requirements	<i>List of requirements that this use-case implementation satisfies</i>
Risks	<i>List of risks for this use-case realization</i>
Development Status	<i>Draft LCO Draft LCA Draft IOC LCO LCA IOC Accepted Build #</i>
User Interface	<i>Pictures and/or descriptions of user interface, if applicable, that is needed to describe the behavior (e.g. reference or URL to prototype screen).</i>
Pre-conditions	<i>Description of state of system and participants before use-case performed. (informal text or OCL, or both)</i>
Post-conditions	<i>Description of state of system and participants after use-case performed. (informal text or OCL, or both)</i>
Priority	<i>Description of the priority of the use-case; using the MSCW (MoScow) prioritization scheme: M (Must have), S (Should have), C (Could have), W (Want to have)</i>

For each process implementation, create a subsection numbered 3.3.X with the name of the process implementation in the header. Each subsection should provide the information shown in table 10 and one or more Interaction Diagrams.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Create at least one implementation for each process.
2. Give each process implementation a name that expresses the behavior that it represents.
 - A good name is usually a verb, or verb phrase.
 - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.
3. To facilitate traceability, assign each process a unique number (e.g. Process-Implementation-01 or UCR-01).
4. Use the “just do it” approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Each implementation may require several iterations to get right. “Go with what you know”, then revise the set of implementation and adjust descriptions, as needed.
5. Start an Interaction Diagram by creating one or more instances of each actor that participates in the use-case realized. List the actions specified in the courses of action for the use-case realized along the left margin of the diagram. For each action,

Additional SSAD Guideline

- Identify component(s) of the system that should participate and instance(s) of analysis classes that are needed. (The activity diagram of the use–case should be helpful in identifying the analysis classes are involved.)
 - Identify process(s) that each component should do.
 - Identify operations(s) on the instances of analysis classes need to be performed. (Each operation should be described in the information class' description (SSAD 3.2.X).
 - Show message(s) that need to be exchanged.
6. Each process implementation should be described in enough detail to be testable.

Model Integration Rules:

1. For LCO,
 - Each system process (SSAD 2.3) that is high–risk, architecturally–significant, or particularly complex should have at least one implementation.

A draft Use–Case Realization Description should be filled out.

A draft Implementation Diagram should be created.

- Each actor instance in the diagram should be an instance of an actor associated with the system process (SSAD 2.3).
- Each non-actor instance in the diagram should be an instance of either a software component, or interface in the architecture (SSAD 3.1.6), or an information class (SSAD 3.2).
- Each message in the diagram should be the name of a process, an operation, or an event defined for either a component or interface in the architecture (SSAD 3.1) or an information class (SSAD 3.2).
- Each action described in courses of action for the use–case realized by one or more messages.

2. For LCA,

- Each system process (SSAD 2.3) that is high–risk, architecturally–significant, or particularly complex should have at least one implementation.

Its Use–Case Realization Description shall be filled out.

An Implementation Diagram shall be created.

- Each actor instance in the diagram shall be an instance of an actor associated with the system process (SSAD 2.3).
- Each non-actor instance in the diagram shall be an instance of either a software component or interface in the architecture (SSAD 3.1.6), or a data model (SSAD 3.2).
- Each message in the diagram shall be the name of a process, an operation, or an event defined for either a component or interface in the architecture (SSAD 3.1) or a information classes (SSAD 3.2).

Additional SSAD Guideline

- Each action described in courses of action for the use–case realized by one or more messages.
3. For IOC, each system process (SSAD 2.3) shall have at least one implementation described to the same level of detail as a high–risk, architecturally–significant, or particularly complex implementation at LCO.

3.4 Architectural Styles, Patterns & Frameworks

Representation:

Create a table like Table 11, describing for style, pattern, and framework.

Table 11: Architecture Styles, Patterns, and Frameworks

Name	Description	Benefits, Costs, & Limitations
<i>Name & Identifier</i>	<i>Briefly describe the style, pattern, or framework; provide a reference to its detailed description.</i>	<i>Summarize key benefits, costs, and limitations of using the style, pattern, or framework in this architecture.</i>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Focus on benefits and costs that are relevant to the system's.

Common Pitfalls:

- Specifying implementation–level styles, patterns, or frameworks. For example, the patterns described in the book *Design Patterns* [Gamma 19988] are general implementation–design patterns. These patterns should be described in SSAD 4.3 in Technology–Specific Architecture Modeling.

Model Integration Rules:

- Each style, pattern, or framework should be used to define the architecture's structure (SSAD 3.1) or its information classes (SSAD 3.2).
- The details of each limitation should be captured in the architecture's structure (SSAD3.1), its information classes (SSAD 3.2).

4. Platform/Technology–Specific Model

Recommendations:

See Recommendations in Platform/Technology–Independent Model (SSAD 3).

4.1 Structure

Recommendations:

The subsections of this section described below are intended to be general purpose. Tailor this section based on the size of the system, the chosen representation language, and the architectural style(s).

Model Integration Rules:

- The rationale for the Architecture must be documented in the FRD.

UML Guideline:

Either refine the contents of the “Architecture Design” package or create a UML package with the stereotype <<systemModel>> the name “Technology Specific Model” to hold all models described in the following sections.

Refining the “Architecture Design” package is simpler; but you lose the Technology–Independent Model view. Creating an “Technology Specific Model” package causes some redundancy; but maintains the Technology–Independent Model view, and facilitates tracing from Technology–Independent Model view to Technology–Specific Model implementations. Creating a separate package is particularly useful when creating a product–line based on the same architecture.

4.1.1 Hardware Classifier Model

Representation:

See Representation in the architecture–design Hardware Classifier Model (SSAD 3.1.1).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.
2. If there are no changes to the Hardware Classifier Model specified during architecture design (SSAD 3.1.1) and the same representation is used, then just reference section 3.1.1.
3. Create additional Static–Structure Diagram that show how each component and connector classifier defined during architecture design (SSAD 3.1.1) is implemented by one or more implementation–specific component and connector classifiers. Represent each component and connector classifier as described under Representation in this section. For each implementation–specific component and connector classifier that implementation–independent component and connector classifier, show a dependency relation with the stereotype <<trace>> from the implementation–specific to the implementation–independent component and connector classifier.
4. For other recommendations, see Recommendations in the architecture–design Hardware Classifier Model (SSAD 31.1.).

Additional SSAD Guideline

Common Pitfalls:

- Confusing component instances with component classifiers.
- Including software component classifiers. (They should be documented in SSAD 4.1.6)
- Including actors that are not described in the system context (SSAD 2.1).

Model Integration Rules:

- Each actor described here should be described in the system context (SSAD 2.1).
- LCA: complete descriptions of all hardware component classifiers needed to support high-risk or architecturally-significant behaviors.
- IOC: at least one instance of each hardware component classifier shown here shall appear in the Deployment Model (SSAD 4.1.3).

577 Guidelines:

See the subsection '577 Guidelines' in the architecture-design Hardware Classifier Model (SSAD 3.1.1).

4.1.2 Software Classifier Model

Representation:

See the Representation discussion in the architecture-design Software Classifier Model (SSAD 3.1.2).

For each implementation-specific software component classifier, create a subsection numbered 4.1.6.x with the name of the software connector classifier in the header, under the Software Component Classifiers (SSAD 4.1.6). The body of the subsection should describe pertinent information about the kind of software component.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.
2. Implementation-level stereotypes (e.g. <<ejbEntity>>) can be used to specify details about how a component classifier is implemented using supporting framework. Any implementation-level stereotypes used should ideally be defined in a standardized UML *Profile* (the description of the Profile should be referenced in SSAD 1.3). If you use an implementation-level stereotype that is not defined in a standard, you need to define the stereotype in either the Glossary for System Analysis and Design or as part of a UML Profile which you define. (The description of each Profile used should be referenced in SSAD 1.3)
3. If there are no changes to the Software Classifier Model specified during architecture design (SSAD 3.1.2) and the same representation is used, then just reference section 3.1.2.
4. Create additional Static-Structure Diagrams that show how each component and connector classifier defined during architecture design (SSAD 3.1.2) is implemented by one or more implementation-

Additional SSAD Guideline

specific component and connector classifiers. Represent each component and connector classifier as described under Representation in this section. For each implementation-specific component and connector classifier that implementation-independent component and connector classifier, show a dependency relation with the stereotype <<trace>> from the implementation-specific to the implementation-independent component and connector classifier.

5. For other recommendations, see Recommendations in the architecture-design Software Classifier Model (SSAD 3.1.2).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Including actors that are not described in the system context (SSAD 2.1).
- Using undefined stereotypes on component classifiers.

Model Integration Rules:

- Each actor described here should be described the system context (SSAD 2.1).
- LCA: complete descriptions of all software component classifier needed to support high-risk or architecturally-significant behaviors.
- IOC: at least one instance of each software component classifier shown here shall appear in the Deployment Model (SSAD 4.1.3).

577 Guidelines:

See the subsection '577 Guidelines' in the architecture-design Software Classifier Model (SSAD 3.1.2).

4.1.3 Deployment Model

Representation:

See the Representation discussion in the Deployment Model specified in architecture design (SSAD 3.1.3).

For each implementation-specific hardware component, create a subsection numbered 4.1.7.x with the name of the hardware component in the header, under the Hardware Components (SSAD 4.1.7). The body of the subsection should describe the hardware component.

For each implementation-specific software component, create a subsection numbered 4.1.8.x with the name of the software component in the header, under the Software Components (SSAD 4.1.8). The body of the subsection should describe the software component.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.

Additional SSAD Guideline

2. Implementation-level stereotypes (e.g. <<ejbEntity>>) can be used to specify details about how a component is implemented using supporting framework. Any implementation-level stereotypes used should ideally be defined in a standardized UML *Profile* (the description of the Profile should be referenced in SSAD 1.3). If you use an implementation-level stereotype that is not defined in a standard, you need to define the stereotype in either the Glossary for System Analysis and Design or as part of a UML Profile which you define. (The description of each Profile used should be referenced in SSAD 1.3)
3. For other recommendations, see Recommendations in the Deployment Model, specified in architecture design (SSAD 3.1.3).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Using undefined stereotypes on components.

Model Integration Rules:

- Each software component classifier described here should be shown in the Software Classifier Model (SSAD 4.1.2) and described in the section Software Component Classifiers (SSAD 4.1.6).
- Each actor described here should be described in the system context (SSAD 2.1).
- At LCA, the System Deployment Model shall show the final configuration of implementation-specific hardware and software that participate in high-risk behaviors, and the preliminary allocation of all other components.
- At IOC, the System Deployment Model shall show the final configuration of all implementation-specific components.

577 Guidelines:

See 577 Guidelines in the Deployment Model specified in architecture design (SSAD 3.1.3).

4.1.4 Hardware Component Classifiers

No additional details

4.1.4.1 Component Classifier X

No additional details.

4.1.5 Hardware Connector Classifiers

No additional details

4.1.5.1 Connector Classifier X

No additional details

4.1.6 Software Component Classifiers

No additional details

4.1.6.1 *Component Classifier X*

No additional details.

4.1.6.1.1 Purpose

No additional details.

4.1.6.1.2 Interface(s)

Representation:

See Representation for Interface(s) of architecture–design Software Component Classifier (SSAD 3.1.6.1.2).

For each feature or set of features, create a subsection numbered 4.1.6.1.2.x and the name of the software feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Model Integration Rules:

- Each interface feature described here, should be used in the component’s behavior description (SSAD 4.1.6.1.4).
- At LCA:

If the component is high–risk, then a set of interfaces for the component shall be described.

If the component is not high–risk, then a draft set of interfaces for the component should be described.

- At IOC: the final set of interfaces for the component shall be described.

577 Guidelines:

See 577 Guidelines for Interface(s) of architecture–design Software Component Classifier (SSAD 3.1.6.1.2).

4.1.6.1.2.1 *Feature or Feature Set X*

See *Feature or Feature Set X* for Interface(s) of architecture–design Software Component Classifier (SSAD 3.1.6.1.2).

4.1.6.1.3 Parameters

Describe any parameters of the software component classifier. The parameters need to be set when an instance of the software component classifier is created.

Additional SSAD Guideline

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other components.

Representation:

See Representation for Parameters of architecture–design Software Component Classifier (SSAD 3.1.6.1.3).

Model Integration Rules:

- At LCA: a preliminary set of parameters for the component, if any exist, should be described.
- At IOC: the final set of parameters for the component shall be described. (If none, say so.)

577 Guidelines:

See 577 Guidelines for Parameters of architecture–design Software Component Classifier (SSAD 3.1.6.1.3).

4.1.6.1.4 Behavior

Describe behavior of the instances of this component classifier.

Representation:

The representation of behavior is depends on the language used to represent it. The following subsection work for UML–based architecture descriptions and may work for ADL’s.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to SSAD 5.

Model Integration Rules:

- At LCA: a draft behavior for the component shall be described if the component is high–risk.
- At IOC: the behavior for the component shall be described if the component is included in the build.

UML Guidelines:

Describe the processes of this component classifier, including how it works with the other components and the system’s actors, and how it uses the artifacts and information; and if significant, how the component’s behavior depends on its mode.

4.1.6.1.4.1 Processes

Describe the processes of this component classifier. For each process, identify which other components and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this component and each actor during the process.

Additional SSAD Guideline

Representation:

Create a Use-Case Model that describes the system's processes; the actors that participate in each process; and the relations among the processes and the outside actors. Represent the model as a package with the stereotype <<use-case model>> with the package name "System Processes".

- Create one or more *Use-Case Diagrams* that show the capabilities, the processes that implement them, and a *realization* relation from each process to the capability that it implements. (Some processes may support the implementation of multiple capabilities.)
- If the relation of the specific processes to the capabilities is not obvious, create one or more *UML Use-Case Diagrams* that show the capabilities, the processes that implement them, and a *realization* relation from each process to the capability that it implements. (Some processes may support the implementation of multiple capabilities.)
- For each process, create a subsection numbered 2.3.1.x (see example section below) with the name of the process in the header. Include the following in the section.
 - A *Use-Case Description* (Table 1 – LeanMBASE Additional SSAD Guideline)
 - An *UML Activity Diagram* for each high-risk or high-priority process and Table 2, 3, 4 – LeanMBASE Additional SSAD Guideline

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. If the component whose behavior is being described is the implementation of a component defined during architecture design, then
 - If there are no implementation-specific processes, you need not create new diagram(s); just insert a reference to the diagram(s) created for the architecture-design component.
 - If there are implementation-specific processes, create Use-Case Diagram(s) that show the implementation-specific processes and their relations.
 - For each use-cases that was described during architecture-design,

Insert a reference to the description created for the architecture-design. (The details should be the same.)

Create an Activity Diagram only if there is implementation-specific behavior.

2. For additional recommendations, see Recommendations for **Error! Reference source not found.** of architecture-design **Error! Reference source not found.** (SSAD 3.1.6.1.4.1).

Common Pitfalls:

- Spending too much time describing the processes in detailed for low risk components. This problem is more common for Simple Systems and small Composite Systems.

Model Integration Rules:

- Each operation defined in the component's interface (SSAD 4.1.6.1.2) must either be represented by a use-case or represented as an action in a use-case's Activity Diagram.

Additional SSAD Guideline

- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).
- Each component in a process description should be defined in the architecture of the system (SSAD **Error! Reference source not found.**).
- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information for the system's architecture (SSAD **Error! Reference source not found.**).

4.1.6.1.4.1.1 *Process X*

Create a section at this level for each process of the component. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Refer section 3.1.5.1.4.1.1 for details.

4.1.6.1.4.2 Modes of Operation

Describe the modal behavior of the component instances, i.e. how the behavior of the component depends on the mode it is in. Describe the high-level states of the component, the *events* that cause the component to change modes, and how the processing is different in each mode.

Representation:

Create a State Model that describes the component's modes in the same way that a model was created to describe the system's modes in SSAD 2.3.2. Represent the model as a package with the stereotype <<state model>> with the package name "Modes of *Component Classifier Name*". Create one or more Statecharts that show the modes of component and the events that cause mode changes.

For each mode, create a subsection numbered 4.1.6.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list component's processes (SSAD 4.1.6.1.4.1) that are available in that mode (see table 8). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode than in another, the process may have mode-specific behavior).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the SSAD 5.

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the <<component>> classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named "State/Activity Modelx" that is attached to the <<component>> classifier, and a diagram named "NewDiagram" in the model. Rename the State Model to "Modes of Operation" and the diagram to "Top Level".

Recommendations:

1. If the component whose behavior is being described is the implementation of a component defined during architecture design, then

Additional SSAD Guideline

- If there are no implementation-specific modes, you need not create new diagram(s); just insert a reference to the diagram(s) created for the architecture-design component.
- If there are no implementation-specific modes or implementation-specific processes defined in SSAD **Error! Reference source not found.**, you need not create new table of processes in modes, just insert a reference to the table created for the architecture-design component.

For additional recommendations, see Recommendations for system's Modes of Operation (SSAD 2.3.2).

Common Pitfalls:

- Spending too much time describing the modes in detail for low-risk components or components that do not have significant modal behavior. This problem is more common for Simple Systems and small Composite Systems.
- For additional pitfalls, see Common Pitfalls for system's Modes of Operation (SSAD 2.3.2).

Model Integration Rules:

- Each component process (SSAD 4.1.6.1.4.1) should be listed as available in one or more modes.
- Each process listed as available in one or more modes should appear in described in SSAD 4.1.6.1.4.1.

4.1.6.1.4.2.1 *Mode X*

See section 3.1.6.1.4.1.2.1

4.1.6.1.5 Constraints

Describe any constraints on the use and implementation of this component that are not captured in other sections of the component description. The following paragraphs describe some typical constraints.

- Rules that the component must implement in support of rules that the system must satisfy;
- Constraints imposed on component's implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on component's implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the component's design and implementation, without getting into implementation details.

Representation:

See Representation for **Error! Reference source not found.** of architecture-design Software Component Classifier (SSAD 3.1.6.1.6).

Recommendations:

1. To facilitate traceability, assign each rule a unique number (e.g. Component-Rule-01 or CR-01).

Additional SSAD Guideline

Common Pitfalls:

- Describing a rule that is already captured in another view of the component (e.g. modes of operation (SSAD 4.1.6.1.4.2)).

Model Integration Rules:

- Each process described in a constraint, should be described in the component classifier's Processes (SSAD 4.1.6.1.4.1).
- Each mode described in a constraint, should be described in the component classifier's Modes of Operation (SSAD 4.1.6.1.4.2).
- Each artifact or information classifier described in a constraint, should be described in the architecture's Information Classes (SSAD 4.1.9).
- IOC: Each component rule should be implemented by one or more elements described in either this component's Internal Architecture (SSAD 4.1.6.1.7), or its TSM (SSAD **Error! Reference source not found.**).

4.1.6.1.6 Internal Architecture

Describe the architecture of this software component classifier that is independent of the implementation technology.

- For components that represent lowest-level modular, deployable, and replaceable part of a system (e.g. the representation of an executable, a link-library, a Java Bean), describe the objects and classes that are used to create the component.
- For other components, describe the subcomponents of this component, what they are expected to do, how they are connected, and what they communicate.

Representation:

Apply the same guidelines to describe the architecture of this component as was used to describe for the system architecture in section TSM (SSAD **Error! Reference source not found.**), i.e. create subsections: "Structure", "Information Classes", "Behavior", and "Patterns & Frameworks".

For components that represent lowest-level modular, deployable, and replaceable part of a system, the internal architecture consists of the objects and their classes that are used to build the component.

- Create a Static-Structure Model that shows the objects and classes that are used to create the component. Create one or more Static-Structure Diagrams that show the resident classes and interfaces that implement this component; the attributes of each resident class; the operations of each resident class; the classes or interfaces defined in other components which are used by the resident classes; and the relations among the classes and interfaces.

Create one or more Collaboration Diagrams show the particular configurations of the class instances ("objects"), possibly for a specific purpose, that implement this component. If two instances interact in this configuration, then show a *link* connecting the two instances.

Each Static-Structure Diagram and Collaboration Diagram should be accompanied by a brief description of its purpose.

Additional SSAD Guideline

(For many systems, a single Static-Structure Diagram and a single Collaboration diagram are sufficient.)

For other components, create a Structure Model that describes the sub-components and their classes as defined in SSAD 4.1

Recommendations:

1. For a System of System, instead of documenting the internal architecture here, include a reference to the project documents (e.g. OCD, SSAD) for the project that is responsible for creating the subsystem represented by this component.
2. If the describing the objects and classes that are used to build the component, create additional Static-Structure Diagrams that show which information classes (SSAD 3.2) is implemented by one or more implementation classes in this component. Represent each analysis class as described in Information Classes (SSAD 3.2), and each implementation class as in this section. For each class that implements an analysis class, show a dependency relation with the stereotype <<trace>> from the implementation class to the analysis class.
Error! Bookmark not defined..Error! Bookmark not defined.
3. Implementation-level stereotypes can be used to specify details about how a component classifier or a class is implementation using supporting framework (e.g. <<JavaBean>> for a component implemented as a Java Bean, or <<JSP>> for class that represents a Java Server Page). Any implementation-level stereotypes used should ideally be defined in a standardized UML *Profile* (the description of the Profile should be referenced in SSAD 1.3). If you use an implementation-level stereotype that is not defined in a standard, you need to define the stereotype in SSAD 5 or as part of a UML Profile which you define. (The description of each Profile used should be referenced in SSAD 1.3)

577 Guidelines:

Create a UML package with the stereotype <<Structure Model>> the name “Architecture” in the package representing this component (i.e. the one with stereotype <<component>> and the name “*Component Classifier Name ID*”) to hold the internal architecture description.

4.1.7 Hardware Components

No additional details.

4.1.7.1 Component X

No additional details.

4.1.7.1.1 Purpose

No additional details.

4.1.7.1.2 Classifier

No additional details.

4.1.8 Software Components

No additional details.

4.1.8.1 *Component X*

No additional details.

4.1.8.1.1 Purpose

No additional details.

4.1.8.1.2 Classifier

No additional details.

4.1.9 Information Classes

Model Integration Rules:

- At LCA: Each implementation class that resides in a high-risk component (SSAD 4.1.9.1.7) shall be full defined.
- At IOC: All implementation classes that reside in all software components (SSAD 4.1.9.1.7) in the Software Classifier Model (SSAD 4.1.2), that reside in high-risk components shall be full defined.

4.1.9.1 Information Class X

No additional details.

4.1.9.1.1 Purpose

No additional details.

4.1.9.1.2 Defined In: Component Name

No additional details.

4.1.9.1.3 Interface(s)

Representation:

Describe the visible features of the implementation class by creating Create the Static-Structure Diagram named "Interfaces".

- Add a classifier for the implementation class.

Additional SSAD Guideline

- Add a classifier with the stereotype <<interface>> and the name of the interface for each set of related operations. Define the operations they contained and show any inheritance relations among the interfaces.
- Connect the classifier representing the implementation class to each interface with a realization relation.

Note: the class' Interface(s) (SSAD 4.1.9.1.3) and Parameters (SSAD 4.1.9.1.4) can be represented using one Static–Structure Diagram (name the diagram “Interfaces & Parameters”). The interfaces and parameters descriptions of multiple classes can be shown on the same diagram. (Place the diagram in the package that holds the component's package.)

For each interface, create a subsection numbered 4.1.9.1.3.x and the name of the interface in the header. The body of the subsection should describe pertinent information about the interface.

Model Integration Rules:

- For each operation interface that a class realizes, the class shall define an operation with the same signature (SSAD 4.1.9.1.7); unless the class is *abstract*.
- Each interface described here, should be used in the class's behavior description (SSAD 4.1.9.1.7).
- At LCA:

If the class is high–risk, then a set of interfaces for the class shall be described.

If the class is not high–risk, then a draft set of interfaces for the class should be described.

- At IOC: the final set of interfaces for the class shall be described.

4.1.9.1.3.1 Interfaces X

Representation:

Refer section 4.1.9.1.3

4.1.9.1.4 Parameters

Representation:

Create a Static–Structure Diagram that shows the implementation class. Add a dashed rectangular box to upper–right hand corner of the classifier. List the parameters in the box.

Note: the class' Interface(s) (SSAD 4.1.9.1.3) and Parameters (SSAD 4.1.9.1.4) can be represented using one Static–Structure Diagram (name the diagram “Interfaces & Parameters”). The interfaces and parameters descriptions of multiple classes can be shown on the same diagram. (Place the diagram in the package that holds the component's package.)

Model Integration Rules:

- At LCA: a preliminary set of parameters for the class, if any exist, should be described.
- At IOC: the final set of parameters for the class shall be described. (If none, say so.)

Additional SSAD Guideline

4.1.9.1.5 Attributes

Representation:

Create a table like Table 12. Describe one attribute of the class in each row of the table.

Table 12: Class Attributes

Name	Type	Multiplicity	Ordering	Initial Value	Visibility	Purpose
<i>Attribute Name</i>	<i>Full name (or name and identifier) of implementation classifier</i>	<i>Optional number of values represented by attribute</i>	<i>ordered unordered (absent)</i>	<i>Expression defining value in class of attribute</i>	<i>Public Protected Private Package (implementation language specific)</i>	<i>Describe purpose of attribute</i>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Give the attribute a name that expresses the mean of its values.
 - A good name is usually a noun (e.g. “color”) or a short noun phrase (e.g. “average duration”).
 - Each name must be unique relative to the owning class. (Two attributes in different classes can have the same name.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.

Common Pitfalls:

- Describing attributes that are never used.

Model Integration Rules:

- Each attribute defined in the class should be used by one or more of the classes operations (SSAD 4.1.9.1.6).
- Each class name in an attribute’s type description should be defined in Implementation Classes (SSAD 4.1.9).

4.1.9.1.6 Operations

Representation:

For each operation, create a subsection numbered 4.1.9.1.6.x with the name of the operation in the header. Each subsection describe the operations purpose, its parameters and result (called a signature), and any pre- and post-condition, as shown in Table 13 (you need not repeat the operator name or identifier).

Table 13: Operation Description

Identifier:	<i>Unique identifier for traceability (e.g. Op-xx)</i>			
Operation Name:	<i>Name of use-case</i>			
Parameters:	<i>A list of parameter descriptions of the form:</i>			
	Kind	Name	Type	Default Value
	[in] out inout	<i>Name of parameters</i>	<i>Full name (or name and identifier) of implementation classifier</i>	<i>Expression defining value in class of attribute</i>
Result:	<i>Full name (or name and identifier) of implementation classifier</i>			
Purpose:	<i>Brief description of purpose</i>			
Pre-conditions:	<i>Description of state that the class instance should be in before operation is performed. (informal text, OCL, or both)</i>			
Post-conditions:	<i>Description the class instance's state after operation is performed. (informal text, OCL, or both)</i>			
Visibility:	Public Protected Private Package <i>(implementation language specific)</i>			
Abstract:	Yes No			
Method:	<i>Overview of the implementation of operation</i>			

If the operation is not abstract, then describe the *method* used to implementation the operation; provide a brief text description and explain any critical algorithms; and for non-trivial operation of a class that is not COTS then provide a detailed description as follows:

- If the operation requests operations of other objects, create a Sequence Diagram to illustrate the interactions;
- If the operation performs some complex logic that is not adequately represented by the Sequence Diagram, then create an Activity Diagram to describe the logic.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Give each operation a name that expresses the behavior that it represents.
 - A good name is usually a verb or verb phrase.
 - Each operation in the class must have a unique name and signature. (Two operations in owned by different classes can have the same name and signature.)
 - Avoid names that sound alike or are spelled alike, as well as synonyms.
 - Clear, self-explanatory names may require several words.

Common Pitfalls:

- Not describing all operations that are used in either use-case realization or a method of this class or another class.

Additional SSAD Guideline

Model Integration Rules:

- Each class name in a parameter's description or operation result should be defined in Information Classes (SSAD 4.1.7).
- At LCA:

The operations for the class that are used in high-risk behaviors shall be fully described, including any Sequence or Activity Diagrams.

- At IOC:

The operations for the class that are used in the build shall be fully described, including any Sequence or Activity Diagrams.

Draft descriptions of other operations should be provided.

4.1.9.1.6.1 *Operation X*

Refer section 4.1.9.1.6

4.1.9.1.7 **State Behavior**

Representation:

Create a State Model that describes the class instance's state behavior. Represent the model as a package with the stereotype <<State Model>> with the package name "States of *Implementation Class Name*". Create one or more UML Statecharts that show the states of class instances, the events (e.g. receipt of operation requests) that cause state changes, and the actions performed by the instance (e.g. request another object perform one of its operations) either when in a state or when changing states.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the class in the Browser View and select New | Statechart Diagram. Rose will create a model named "State/Activity Modelx" that is attached to the class, and a diagram named "NewDiagram" in the model. Rename the State Model to "State behavior" and the diagram to "Top Level".

Recommendations:

1. Describe the state behavior of the class if
 - The class participates in component/system processes that have high safety or high reliability L.O.S. requirements;
 - An instances of the class represents a concurrent element (e.g. a OS process or thread) or represents a resource can be shared by two or more concurrent elements;
 - Some of the class' operations can only be executed when an instance is in a certain state or the details of some methods change depending on the state of the instance.

Additional SSAD Guideline

Common Pitfalls:

- Spending too much time describing the states in detailed for low-risk classes or classes that do not have significant modal behavior. This problem is more common for systems that do not have high safety or reliability L.O.S. requirements.

Model Integration Rules:

- Each operation of this class (SSAD 4.1.9.1.6) should appear in the class' state model as an event for one or more transitions (internal or between states).
- Each operation described as an event for one or more transitions (internal or between states) shall be described in this class' operations (SSAD 4.1.9.1.6).
- Each operation requested in an action or condition shall be described in the operations of the class of the instance to which the request is sent (SSAD 4.1.9.1.6)
- At LCA: a draft behavior for the class shall be described if the class is involved in high-risk behavior.
- At IOC: the behavior for the class shall be described if the class is included in the build.

4.1.9.1.8 Constraints

No additional details.

4.1.9.2 Objects

No additional details.

4.1.9.3 Object X

No additional details.

4.1.9.3.1 Purpose

No additional details.

4.1.9.3.2 Classifier

No additional details.

4.1.9.3.3 L.O.S.

The L.O.S. Goals for an object's information classes defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this object must satisfy the values specified either by the classifier or this instance.

Additional SSAD Guideline

Representation:

Create a table like Table 14, where a value is specified for each goal that applies to this an object’s information class, where the classifier described as “value is specific to the object”.

Table 14: L.O.S. Values for a Classifier Instance

Goal	Value
<i>Goal id & description</i>	<i>Component-specific value(s).</i>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

- Each goal described here, should be described in the **Error! Reference source not found.** of the object’s information class
- LCA: If this object is high–risk, then each goal in the **Error! Reference source not found.** of the object’s information class that is described as “value is specific to the object”, shall have a value.
- IOC: Each goal described in the **Error! Reference source not found.** of the information class described as “value is specific to the object”, shall have a value.

4.2 Behavior

Representation:

See Representation in the Behavior section (SSAD 3.3) of TSM.

For each process implementation, create a subsection numbered 4.2.X, with the name of the process implementation in the header. Each subsection should provide the information shown in table 7 and one or more Interaction Diagrams.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Start an Interaction Diagram by creating one or more instances of each actor that participates in the use–case realized. List the actions specified in the courses of action for the use–case realized along the left margin of the diagram. For each action,
 - Identify component(s) of the system that should participate and instance(s) of implementation classes that are needed. (The activity diagram of the use–case should be helpful in identifying the analysis classes are involved.)
 - Identify process(s) that each component should do. (Each process should be described in the behavior of the component’s class (SSAD 4.1.6.1.4).)
 - Identify operations(s) on the instances of implementation classes need to be performed. (Each operation should be described in the class’ description (SSAD 4.1.9.x).

Additional SSAD Guideline

- Show message(s) that need to be exchanged.
2. For additional recommendations, see Recommendations in the Behavior section (SSAD 3.3) of architecture design.

Common Pitfalls:

See Common Pitfalls in the Behavior section (SSAD 3.3) of TSM.

Model Integration Rules:

- For LCA,

Each system process (SSAD 2.3.1) that is high-risk, architecturally-significant, or particularly complex should have at least one implementation.

- A draft Use-Case Realization Description should be filled out.
- A draft Implementation Diagram should be created.
 - Each actor instance in the diagram should be an instance of an actor associated with the system process (SSAD 2.3).
 - Each non-actor instance in the diagram should be an instance of either a software component, or interface in the architecture (SSAD 3.1.6), or the information class (SSAD 4.1.9).
 - Each message in the diagram should be the name of a process, an operation, or an event defined for either a software component or interface in the architecture (SSAD 3.1.6), or an information class (SSAD 4.1.9).
 - Each action described in courses of action for the use-case realized by one or more messages.

- For IOC,

Each system process (SSAD 2.3) that is high-risk, architecturally-significant, or particularly complex should have at least one implementation.

- Its Use-Case Realization Description shall be filled out.
- An Implementation Diagram shall be created.
 - Each actor instance in the diagram shall be an instance of an actor associated with the system process (SSAD 2.3).
 - Each non-actor instance in the diagram shall be an instance of either a software component, or interface in the architecture (SSAD 3.1.6), or the information class (SSAD 4.1.9).
 - Each message in the diagram shall be the name of a process, an operation, or an event defined for either a software component or interface in the architecture (SSAD 3.1.6), or the information class (SSAD 4.1.9).
 - Each action described in courses of action for the use-case realized by one or more messages.

Additional SSAD Guideline

- For IOC, each system process (SSAD 2.3) shall have at least one implementation described to the same level of detail as a high-risk, architecturally-significant, or particularly complex implementation at LCO.

4.3 Patterns & Frameworks

Representation:

See Representation in the Architecture Style's Patterns and Framework section (SSAD 3.4) of architecture design. Include descriptions of features of the style, pattern, or framework description.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

See Recommendations in the Architecture Style's Patterns and Framework section (SSAD 3.4) of architecture design.

Model Integration Rules:

- Each style, pattern, or framework should be used to define the implementation structure (SSAD 4.1).
- The details of each limitation should be captured in the implementation structure (SSAD 4.1).

UML Guideline:

Create a package name "Frameworks". For each framework (e.g. Java), create a package with the name of the framework. In each framework package, create Static-Structure Diagrams that describe any classes, interfaces, or objects that are defined in the framework and are used by the implementation.

577 Guideline:

Follow UML Guideline.

4.4 Project Artifacts

Representation:

Create a hierarchical list of directories, their files, and the classes or components in each file. If all the classes of a component are in the same file, then just list the component name; otherwise list the specific classes. For example:

```
Directory Name 1
  File Name 1
    Component Classifier Name 1
  File Name 2
    Component Classifier Name 2
```

Additional SSAD Guideline

Component Classifier Name 3
File Name 3
Component Classifier Name 4 . Class Name 1
File Name 4
Component Classifier Name 4 . Class Name 2
Component Classifier Name 4 . Class Name 3
Directory Name 2
...
Directory Name N
Subdirectory Name 1
File Name 1
Component Classifier Name 5
File Name 2
Component Classifier Name 6
Subdirectory Name 2
...
Subdirectory Name N

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design.

Recommendations:

1. Include all required files, scripts, programs, images, and libraries in the directory structure.
2. Do not describe details of COTS products.
3. One strategy for defining the directory structure is to parallel the package hierarchy in the Software Classifier Model (SSAD 4.1.2). Another strategy is to create packages that reflect different categories of artifacts. Mixed strategies are common.

Model Integration Rules:

- At LCA, the Configuration Model should show the configuration of any high-risk components and classes.
- At IOC, the Configuration Model shall show the configuration of all components and classes.

UML Guideline:

Create a package with the label “Configuration Model”. In this package, create one or more UML Component Diagrams that show the project artifacts; the project directories; the contents of each project artifact; and the dependency relations among project artifacts and directories. Represent each project artifact as a classifier icon with one of the UML stereotypes shown in table 15, or with an implementation- or platform-specific stereotype (e.g. Webpage, jarFile, script).

Table 15: Stereotypes for Project Artifacts

General	
<<file>>	a physical file that is otherwise undifferentiated
<<table>>	a database table
Specialized Files	
<<document>>	a generic file that is not a «source» file or «executable»
<<executable>>	a file that can be executed on a computer
<<library>>	A static or dynamic library file.
<<source>>	a compilable source code file
<<document>>	a generic file that is not a source or executable file

Create a Component Diagram that shows the high-level directories. Represent each directory used to organize the project artifacts as a package. In each package, create a Component Diagram that shows the contents of the directory represented by the package.

If the contents of one package depend on the contents of another package, add a dependency relation from the client package to the provider. If a project artifact depends on another project artifact or the contents of a package, add a dependency relation from the client to the provider.

For each artifact, create a UML Component diagram in the package containing the project artifact that shows the project artifact and any components or implementation classes that are in the project artifact. If a project artifact contains one or more components, then either nest each component in the project artifact, or show an aggregation relation from the artifact to each component in the artifact. If an artifact contains only a subset of the classes of a component, then either nest each class in the project artifact, or show an aggregation relation from the artifact to each implementation class in the artifact.

Multiple artifacts and their contents can be shown on one diagram, if the diagram is readable and clear. For example, create a diagram that shows all project artifacts that are webpages, and another that shows all project artifacts that are database files. It is often necessary to include a one or more artifacts from another package on a diagram that is otherwise devoted to a particular package in order to show dependencies on those artifacts.

Rational Rose Guideline:

Rose's Component Diagram only allows dependency relations between diagram elements, and does not allow classifiers on the diagram. Either:

1. Follow the directions given in the UML Guidelines, except create Static-Structure Diagrams instead of the Component Diagrams, and represent each project artifact as a classifier with one of the UML stereotypes shown in Table, or with an implementation- or platform-specific stereotype (e.g. Webpage, jarFile, script).
2. Follow the directions given in the UML Guidelines, except

Instead of creating package with the label "Configuration Model", use the predefined Rose package labeled "Component View";

Additional SSAD Guideline

Create Component Diagrams that show the packages and project artifacts (do not forget the stereotypes);

Open the Specification for the project artifact, switch to the “Realizes” tab, and select the classifiers that are contained in the project artifact. (A potentially faster way to assign a classifier to a component is to drag the classifier from the browser view and “dropping” it on the project artifact on a Component Diagram.)

The first approach results in a model that closely resembles the UML Specification. The second approach, using Rose’s Deployment View, has the advantage that ROSE’s code generator will use the information in the Component View when generating code.

577 Guideline:

Follow ROSE Guidelines for using Rose’s Component View (#2).

5. Glossary for System Analysis and Design

No additional details.

6. Appendices

No additional details.