ACTL Systems Ltd.
217/5 Jaffa St. P.O.B. 8129
91081 Jerusalem, ISRAEL
Tel. 972-2-5376459
Fax. 972-2-5370425

Process Summary for

Agile OO Requirements Capture, Analysis and Design

Updated March 2008

Version 4.0

Doc. Name : ..\ Courses_New\OO-Process overview (UML 2.0).doc
Author : Dani Mannes
Date : 10/03/2008

Copyright

This material has been prepared by:

ACTL Systems Ltd.

217/5 Jaffa St. , POB 8129

Jerusalem 91081

Israel

Copyright © 1996 - 2010, ACTL Systems Ltd., all rights reserved

The content of this manual is the property of ACTL Systems Ltd. and is protected by Israel copyright laws, and various international treaties.

This material is intended for personal, non-commercial use. You may not modify, publish, participate in transfer or sale of, reproduce, scan, create new works from or based on, distribute, perform store on any magnetic device or a retrieval system, transmit in any mean, display, record any trademark copyright or other notice from copies of the contents. You may not use the material for the purpose of training of any kind, internal or for customers, without beforehand written approval of ACTL Systems Ltd.

Table of contents

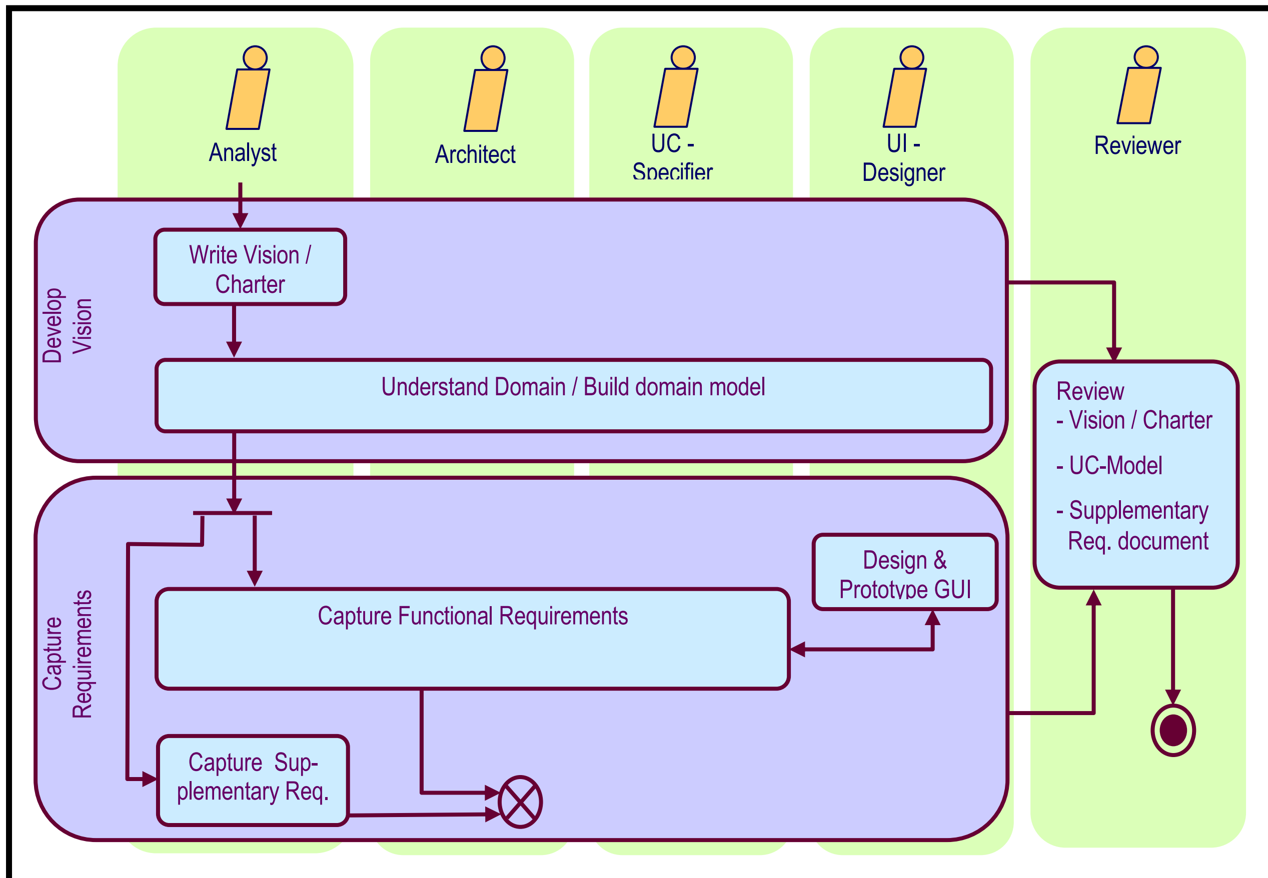
1. Requirements Capture 4
1.1 Develop a vision 4
1.1.1 Write a System Charter 4
1.1.2 Work on Domain Model 5
1.2 Capture Requirements 5
1.2.1 Capture Functional Requirements (=> Create Use Case Model) 5
1.2.2 Capture Supplementary Requirements 5
2. Analysis 6
2.1 Architectural analysis: 6
2.1.1 Identify key abstractions (Classes) 6
2.1.2 Describe remaining classes through 7
2.1.3 Work on class relations 7
2.2 Use Case analysis: 8
2.3 Consolidate and Review Analysis Model 8
3. Design 9
3.1 Logical Architecture 9
3.1.1 Reduce class dependencies 9
3.1.2 Define packages and build component view 10
3.1.3 Model system interfaces 11
3.2 Physical Architecture 12
3.2.1 Build process view 12
3.2.2 Build deployment view 12
3.2.3 Design Components 12

 © Copyright by ACTL Systems Ltd.

1. Requirements Capture

The goal of Requirements Capture is to understand, prioritize and document what the customer really needs and to define the system scope and boundaries.

The following activity diagram shows the process of how to reach this goal. Two workflows; "Develop a Vision" and "Capture Requirements" cover the requirements capture discipline.



1.1 Develop a vision

The goal of this workflow is to get an overall picture of the system to be build. Perform the following activities:

1.1.1 Write a System Charter

Goal: Provide an overview of the Project and allow Management to decide whether to continue or not (Go /NoGo). It is also a kind of letter of intention.

Structure:

- **Introduction**: Executive summary of the System Charter.
- **Goals**: Answers the question WHY the defined FR & NFR are delivered.
- **Funct. Req.**: A short description of what the system will do.
- **Non Funct. Req.**: A short description of the constraints that apply to the system as well as of the environment, users and other non functional issues.

 © Copyright by ACTL Systems Ltd.

1.1.2 Work on Domain Model

Goal: Allow the customer and the development team to synchronize their understanding of the problem domain.

This model helps the development team to formulize its understanding. It doesn't contain any analysis conclusions. It is an informal model that does not need to be maintained!

1.2 Capture Requirements

The goal of this workflow is to capture the functional as well as the supplementary requirements of the system. A requirement is either something the stakeholder asks for, something the team has a need of or something that results from other requirements. In any case a requirement does not represent a solution!

1.2.1 Capture Functional Requirements (=> Create Use Case Model)

Goal: Describe how the user is going to use the system as opposed to describing the systems functionality.

The Use Case Model fully specifies all functional system requirements. A Use Case describes a complete course of interactions taking place between an initiating actor, the system and other participating actors. It produces an observable result, having a value for the initiating or any other participating actor.

Process:

1. Find actors (start with the initiators) and describe them (Definition, Role, Responsibility)
2. For each actor find the use cases he may initiate.
3. Draw an initial use case diagram.
4. Specify goal & description for a subset of 2 (the architect performs UC selection).
5. Specify pre & post conditions for a subset of 4.
6. Specify basic flow, exceptional flows, alternate flows of 5. Use activity and sequence diagrams to visualize your flows.
7. Rework UC-model e.g. draw several diagrams each focusing on a group of use case such as core, utilities, services or any other classification. This grouping may be shown with packages.
8. Structure the use case model by identifying <<extend>>, <<include>> relations or generalizations between use cases.

1.2.2 Capture Supplementary Requirements

Goal: Capture additional requirements and constraints that have an impact on how to construct the system.

Supplementary requirements can be classified as:

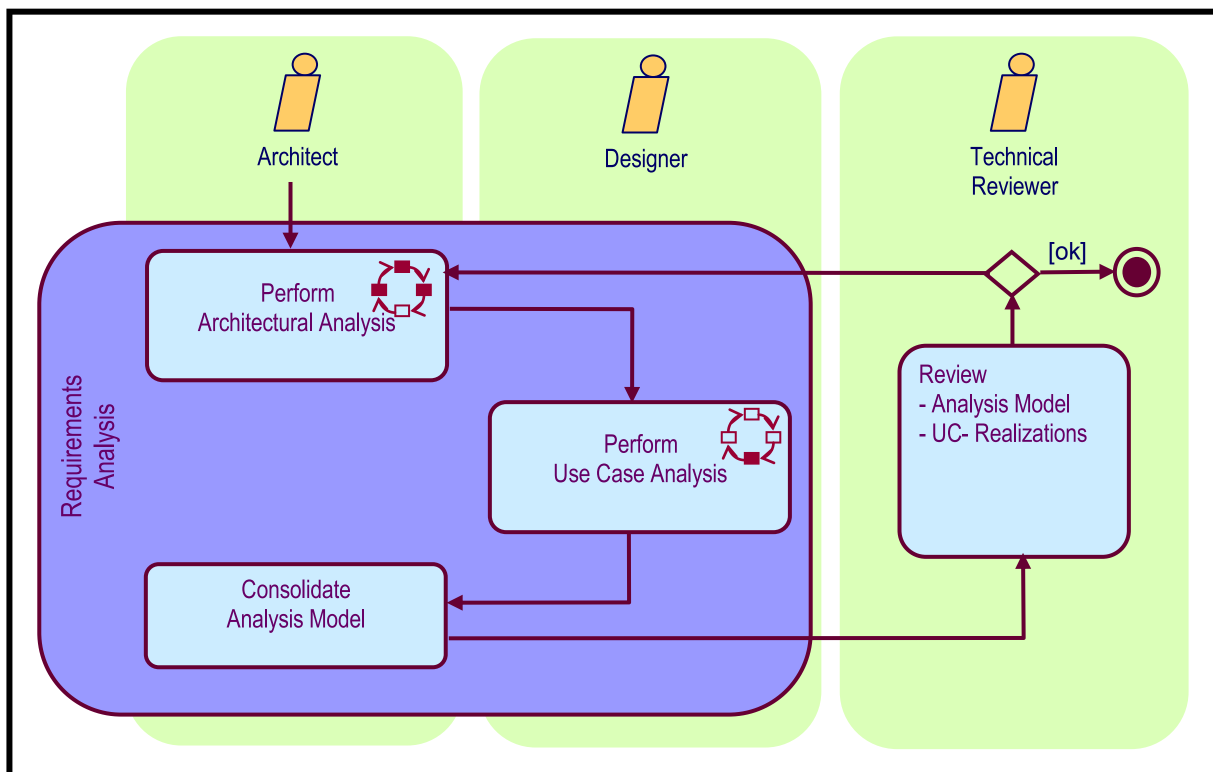
Requirements	Constraints
Usability	**D**esign
Reliability	**I**nterface
Performance	**P**hysical
Supportability	**I**mplementation

 © Copyright by ACTL Systems Ltd.

2. <u>Analysis</u>

The goal of analysis is to find a logical solution to all Use Cases(FR). This solution is documented in an analysis model whose static view is shown with class diagrams and whose dynamic view is shown using sequence and communication diagrams as well as through state machine diagrams.

The requirements analysis workflow has three activities: “Architectural Analysis”, “Use Case Analysis” and “Consolidation of the Analysis Model”. The micro process is used iteratively during the first two activities.



2.1 Architectural analysis:

<u>Goal:</u> Define a candidate architecture in terms of classes and their relations and stereotypical interactions in order to set the stage for the subsequent activity; “Use Case Analysis”.

2.1.1 Identify key abstractions (Classes)

<u>Process:</u>

1. Create a list of class candidates using one of the following techniques:
   - Text analysis -> nouns, or noun phrases become class <u>candidates</u>.
   - Brain storming sessions -> document classes using CRC-cards.
   - Looking for: Places, interactions, roles, organizations, events, people,…..
   - Using experience, analysis patterns, frameworks
   - Use case realizations
2. Filter the candidates list to ~ 50 by eliminating candidates if:
   - Two candidates represent the same concept

- The candidate represents a too general concept
- The candidate is an attribute or operation of another candidate
- The candidate is derived from another candidate
- The candidate is just irrelevant
- The candidate describes an implementation issue
- The candidate is not in scope
- ….

Eliminated class candidates are not deleted. They serve as a repository of class names during analysis, design, implementation and later increments.

3. Draw an initial class diagram

   No emphasis on associations specifications or inheritance **yet**. The purpose of this diagram is to get a graphical representation of the "surviving" classes. Often as a result of the drawing, additional candidates can be eliminated.

2.1.2 Describe remaining classes through

- ***Definition*** describes the class in terms of what it represents.
- ***Role*** describes what the class is doing for other classes.
- ***Responsibility*** describes what the class is doing without being asked to.

2.1.3 Work on class relations

Process:

1. Elaborate the analysis model by focusing on the associations.
   - Ask if an association between classes are missing or superfluous.
   - Label the association.
   - Specify multiplicity.
   - Specify aggregation semantics.
   - Consider if aggregation is strong => Composition.
   - Consider to replace label by Role Names.
   - Provide textual description of the association specifying constraints and rules.
   - Consider to add association classes.
2. Elaborate the analysis model by working on generalization relations and assure that:
   - Each generalization represents an "is_a" and not only an "is_kind_of" relation.
   - Generalization trees are balanced.
   - The derived classes extend and not restrict the base classes.
   - The substitution principle is satisfied.
   - Behavioral compatibility between derived and base classes is assured.
   - Delegation was considered as an option.
   - The right generalization was modeled (e.g. do 2 derived classes relate to the same physical reality, or can an object change to another object during its lifetime?)

 © Copyright by ACTL Systems Ltd.

2.2 Use Case analysis:

Goal: Prove that the static model (elaborated during architectural analysis) is capable of satisfying the functional requirements. => model the dynamic system aspects.

Process:

1. Create a use case realization for each use case.
2. Supplement use case descriptions with eventually missing details required in order to offer a solution.
3. For each use case realization, draw at least one class diagram showing the classes needed to realize the use case. Then draw at least one sequence diagram holding an object for each of the required classes. This sets the stage for the next step.
4. Find operations for the classes involved in the use case. Do this using the following approach:
   - Create modeling mass by "inventing" operations for the classes. You do this by reading the class description and deriving operation from the class's role and responsibility.
   - Draw sequence diagrams showing the required interaction, in order to satisfy the main flow as well as the critical exceptional and alternate flows. In this process you will use operations identified in the previous step, but you will also identify many missing operations.
   - Focus again on the individual classes and identify additional missing operations.
   - Model the state dependent behavior of the classes by drawing state machine diagrams. During this process additional operations may be identified.
5. Complete class descriptions and assure that the operations fit the role and responsibility description of the class.
6. Describe operations and identify class attributes.

2.3 Consolidate and Review Analysis Model

Process:

1. Consolidate the analysis model

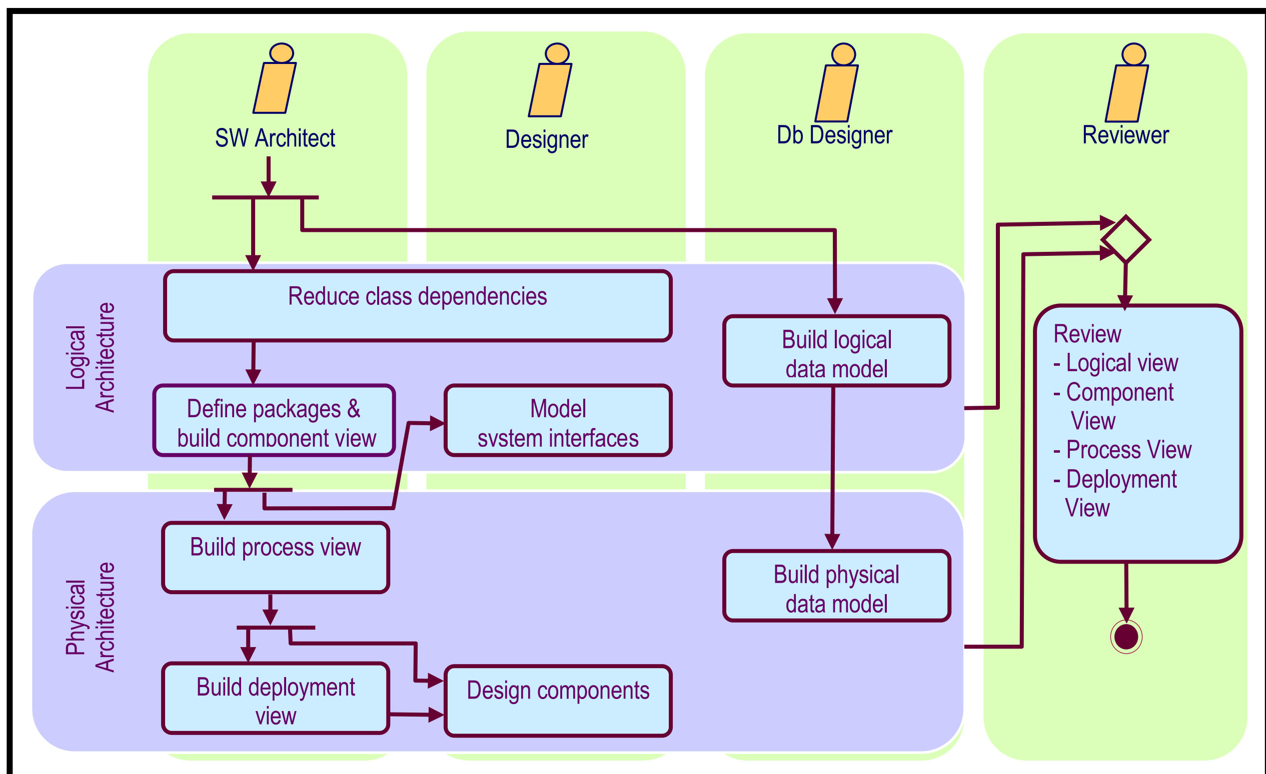
   As several designers could have worked concurrently on several use case realizations, it may be that new classes have been found or the same classes have been modified. Therefore:
   - Collect new classes and integrate them into the global analysis model
   - Merge duplicate classes describing the same concept.
   - Update the use case realizations accordingly.
2. Review the analysis model:
   - Does each class have a definition, role, responsibility?
   - Does each class have at least one operation?
   - Does each diagram have a documentation?
   - Is each operation documented?
   - Has each association a multiplicity?
   - Are all selected UC implementable with the existing classes, operations, relations?

 © Copyright by ACTL Systems Ltd.

3. Design

The goal of design is to create a model that can be implemented so that the resulting system satisfies all functional as well as supplementary requirements selected for the current increment. The static view of the model is described through class, component composite structure and deployment diagrams. The dynamic model is described through interaction, timing and activity diagrams and state machine diagrams.

Design has two workflows; "Logical Architecture" and "Physical Architecture"



3.1 Logical Architecture

The goal of building the logical architecture is to define the large scale structures and system interface mechanism that underpin system realization. This is achieved in a three step approach: reduce class dependencies then define the large scale structures (packages and components) and finally define the system interface mechanisms.

3.1.1 Reduce class dependencies

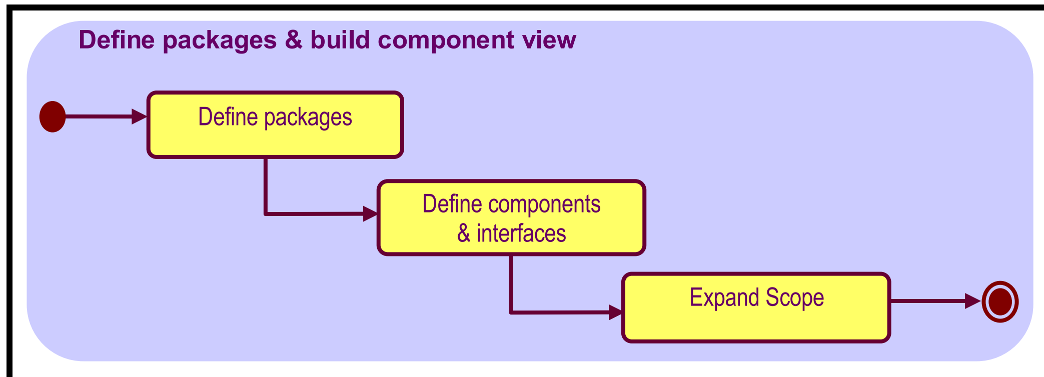
Reducing class dependencies is the cardinal in order to build a good logical architecture. Inspect the class model by checking the following 3 design principles:

- Open Closed Principle
- Liskov Substitution Principle
- Dependency Inversion Principle

 © Copyright by ACTL Systems Ltd.

3.1.2 Define packages and build component view

This activity is the key activity of the logical architecture workflow. It is composed of three actions.



3.1.2.1 Define packages

During Package Design, the analysis model is partitioned into packages using the following architecture design principles:

- **Principle of Reuse**
  Only packages are reused and therefore not single classes are released.
- **Principle of common closure**
  If one class of a package is changed most probably all other classes will need to be changed as well.
- **Principle of Package dependency**
  The package diagram should be a directed acyclic graph.
- **Principle of dependency direction**
  Dependency between packages should run in the direction of stability.
- **Principle of package stability**
  The more stable a package is, the more abstract classes it should contain.

3.1.2.2 Define components and interfaces

The component view is defined in terms of components, their interfaces and component wiring. You build the component view as follows:

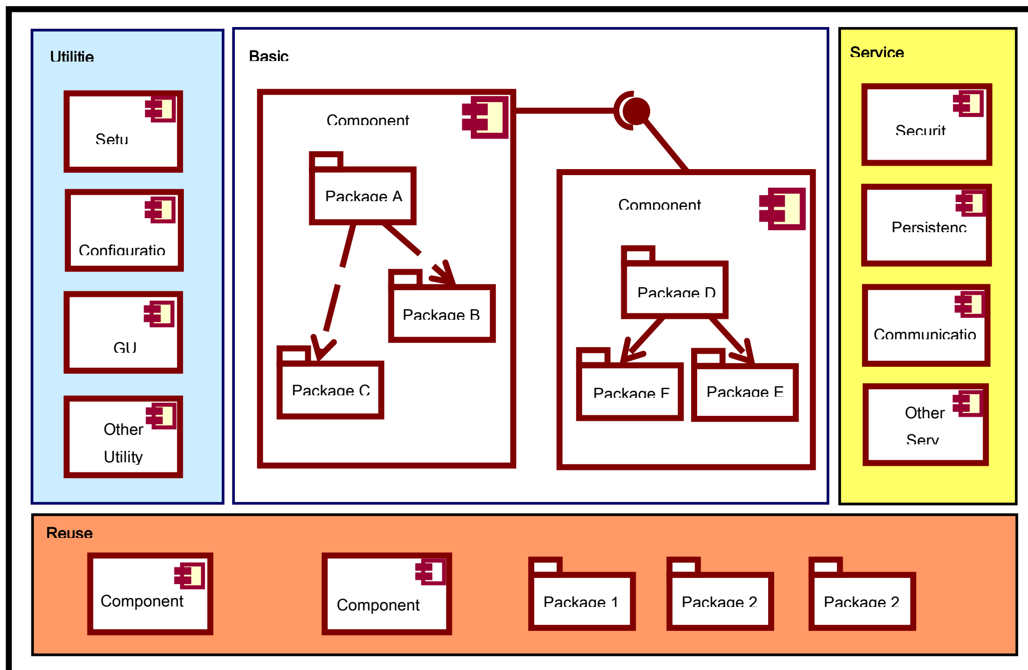
1. Identify components and interfaces and then draw component diagrams.
   Components result from either "upgrading" classes to components, or by grouping a number of classes/packages into a component.
2. Synchronize the package architecture with the component architecture.
3. Show use case realizations on the component level.

 © Copyright by ACTL Systems Ltd.

3.1.2.3 Expand system scope

Issues left open during requirements capture, analysis and logical design need to be resolved. Therefore:

- Review the current model for completeness (attribute, operation , association, generalization, class specification)
- Look for missing class dependencies
- Identify significant missing supplementary requirements
- Extend system scope by considering required utilities, services and reusable assets.



3.1.3 Model system interfaces

This is the last step of logical design. Classes representing "actors" are the roots for system interfaces. Use design patterns for this task.

- For modeling Human Machine Interfaces use the Model View Controller (MVC) pattern.
- For modeling Machine Machine Interfaces use the Proxy pattern.

 © Copyright by ACTL Systems Ltd.

3.2 Physical Architecture

The goal of physical architecture is to extend the model so that it satisfies all requirements including the supplementary requirements (URPS, DIPI) and to address issues of concurrency, communication mechanisms, component initialization, artifact definition and artifact deployment.

3.2.1 Build process view

The process view models the concurrency aspects of the system. It focuses on the active classes, their internal structure, communication mechanisms and synchronization. You build the process view as follows:

1. Identify processes and threads and model their internal structures and show them in a class diagram that only shows them.

   Active classes result from either "upgrading" existing classes or by "wrapping" existing classes by new ones that are active.
2. Model thread / process communication and synchronization using timing, activity and other useful diagrams.
3. Define & document communication mechanism and model them using UML diagrams.

3.2.2 Build deployment view

The deployment view reflects the physical system architecture in terms of artifacts and nodes. You build the deployment view as follows:

1. Identify supplementary requirements affecting the deployment view.
   - Usability, Reliability Performance and Supportability requirements.
   - Design and Physical constraints.
2. Model the deployment view by drawing one or more deployment diagrams that show the nodes and their associations.
3. Define node communication mechanisms.
4. Enhance the deployment view with the artifact deployments and deployment specifications.

3.2.3 Design Components

When designing the internals of a component you have to design the following aspects:

1. Port and interface realization.
   - For components whose internal structure is already known, use composite structure diagrams showing the wiring of ports and interfaces to the internal parts.
   - For components that are "upgraded" classes, a "new" project" has just been identified. You therefore have to capture requirements and perform analysis and design.
   - In any case good component documentation also uses dynamic diagrams such as sequence diagrams to show the internal mechanisms of the interface realization.
2. Component initialization.
   - Draw composite structure diagrams for each component depicting which parts need to be created upon component initialization
3. Component manifestation.
   - Identify artifacts and draw deployment diagrams for each component that show all the manifesting artifacts.
4. Design internal component mechanisms with sufficient details so that it can be realized.

 © Copyright by ACTL Systems Ltd.