Rational System Architecture

December 15, 1986

Table Of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to provide a preliminary description of the long term system architecture for Rational products. After further review and development, a more detailed description will be written.

1.2. Background

This architecture is designed to address various functional, operational, economic and competitive issues, some of which are explicitly addressed in the closing section of this document. However, the fundamental impetus for this architecture is the customer's software engineering process and the evovling ability of the Rational environment to support this process.

Most Rational customers are developing systems that are between 50K SLOC and 2M SLOC (SLOC = source line of code). Some of the larger programs (SDI, Space Station, WIS, etc.) are larger than 2M SLOC, but are usually broken up into several procurements of less than 2M SLOC.

Medium and large systems are broken up into manageable components (CSCI's) that are usually less than 40K SLOC. These components roughly correspond to one or a few Rational subsystems.

Development of a small number (often one) of these components is usually assigned to a small development group (2-20 primary developers). The various software groups are responsible for design, code, and unit test. The software development group works closely with systems engineers who wrote the detail requirements (sometimes system engineers and software engineers are the same people, but not usually) and with a central architecture group responsible for overall system design.

Generally, there is a separate test and integration group which accepts the various software components and performs system test functions. Most programs are moving toward more incremental integration, where early and reasonably frequent (1-6 month interval) system releases are produced with limited functionality to identify integration problems early. Some projects still defer all integration to a single large integration activity. For larger programs there will be several levels of release with specific test and QA steps required to promote software to the next release level.

This development process maps fairly naturally onto the Rational environment, at least it maps nicely onto the Epsilon design in the single machine case. Subsystems provide a fundamental building block and the CMVC mechanisms support project management and the release process. The current Epsilon design has a simpler model for multi-machine development than the one assumed in the system architecture discussed below. Thus the system architecture assumes the existence of an environment release (Zeta) that has not yet been designed. The main differences between Epsilon and Zeta involve support for distributed development of a single subsystem on multiple machines.

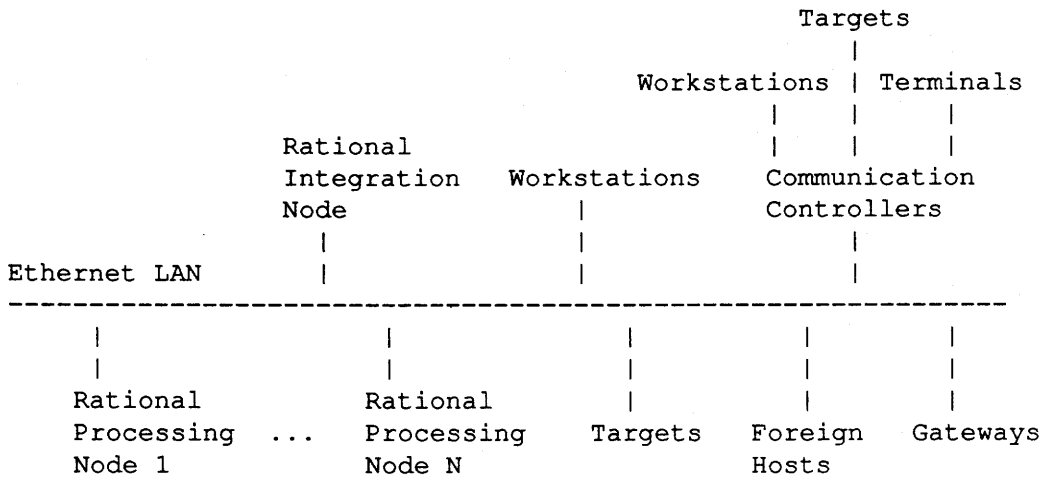
1.3. Scope

This document will first outline the overall topology and structure of the proposed system architecture. Then various compilation and configuration management issues are discussed, followed by editting and debugging issues. Next the architecture is discussed from the point of view of operations and support activities. Then an intermediate form of the architecture is described, based on the capabilities of of Epsilon_0 plus a few minor extensions. Finally, the proposed architecture is evaluated relative to a number of criteria important to the business over the next 3-5 years.

2. Architecture

2.1. Topology

The overall architecture is based on connecting Rational processors to workstations, targets, terminals, foreign hosts, and other Rational processors using high-performance local area network technology. For the next several years this primarily means Ethernet.



2.2. Integration Node

The integration node would be a large configuration with significant disk and tape resources.

Functionally, it would serve as the repository for both the CMVC database and the main or "golden" paths for each subsystem. The integration, test and release, and project management activities would use the integration node (along with some number of processing nodes for test execution).

Operationally, backup & recovery and software distribution activities would occur only on the integration nodes.

2.3. Processing Node

The processing node is a smaller node with no tape or comm resources (other than the ethernet interface) and limited disk capacity. These are the nodes where active design, code, debug, and test activities occur. Thus the active paths and subpaths reside on the processing nodes. Since the CMVC database is on the integration node, loss of a processing node can only lose work that has not been checked in. Additionally, frequent merge with paths on the integration node (which should be very cheap with epsilon mechanisms) means that very little compilation is lost when a processing node is lost.

Development on the processing nodes can proceed when the integration node is down, but the CMVC information must be resynchronized when the integration node is restored. At such time there may be conflicts that cannot be resolved automatically (simultaneous check out on two processing nodes, compatibility conflicts, object creation conflicts), requiring some manual action.

The processing nodes are intended to require minimal care and feeding. No backup and recovery activities take place on the processing nodes. CMVC operations should be able to (inexpensively) guarantee that all interesting information is on the integration node. Software releases are loaded onto

the integration node and distributed over the network.

Processing nodes may also be used by the integration and test groups to run system tests and perform other integration activities.

2.4. Workstations

2.5. Terminals

2.6. Targets

2.7. Foreign Hosts

2.8. Gateways

2.9. Communications Controllers

3. Compilation and Configuration Management

4. Editting and Debugging

5. Operations

6. Intermediate Capability

7. Evaluation

Range of Project Size and Structure
Increment of Purchase and Low Cost
Ease of Use
Ease of Operation
Flexibility
Feasibility & Implementability