

Creating End-to-End Middleware Services Via Configuration

Sanjai Narain

narain@research.telcordia.com

Kirthika Parmeswaran

kirthika@research.telcordia.com

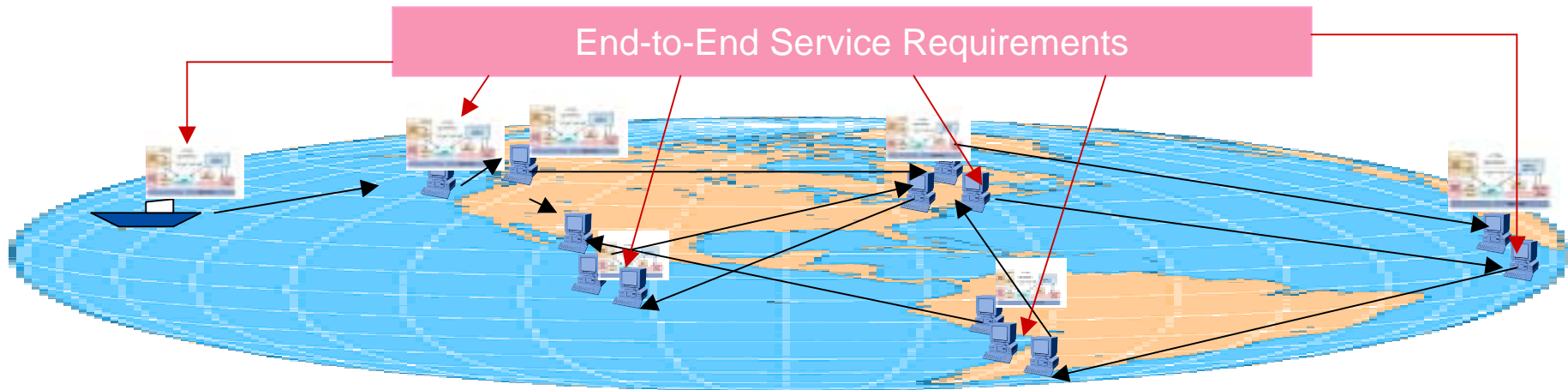
Pradeep Gore

pradeep@oomworks.com

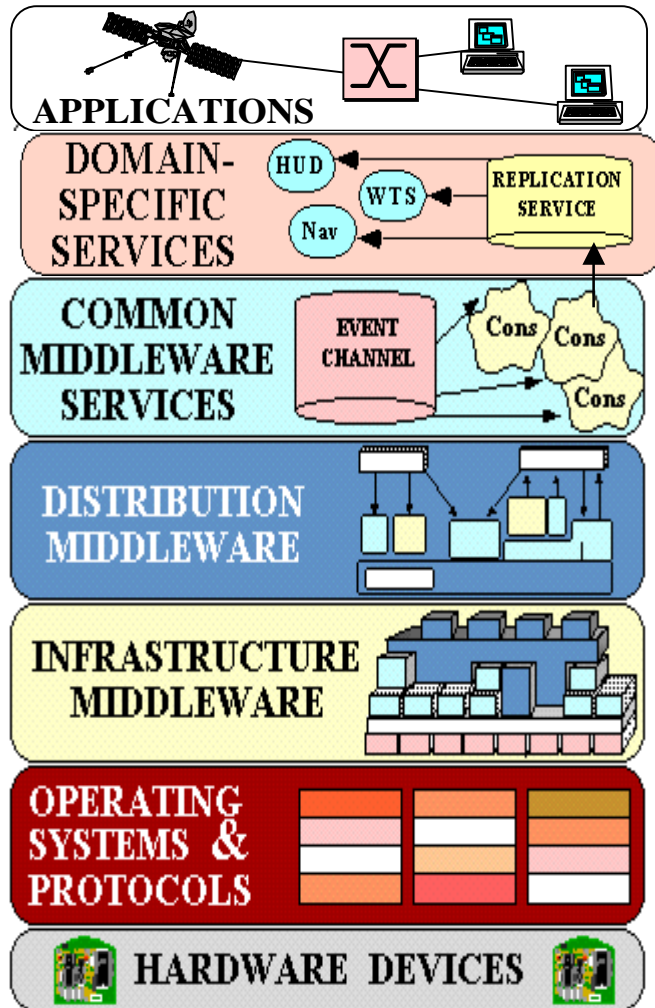
Real-Time And Embedded
Distributed Object Computing Workshop
July 15-18, 2002

Service Creation Challenges for Large Scale DRE Systems

- Service must function across diverse domains and platforms with stringent QoS, real-time guarantees
- Service piece-parts must be logically integrated into an end-to-end service architecture
- Service provisioning should be automated

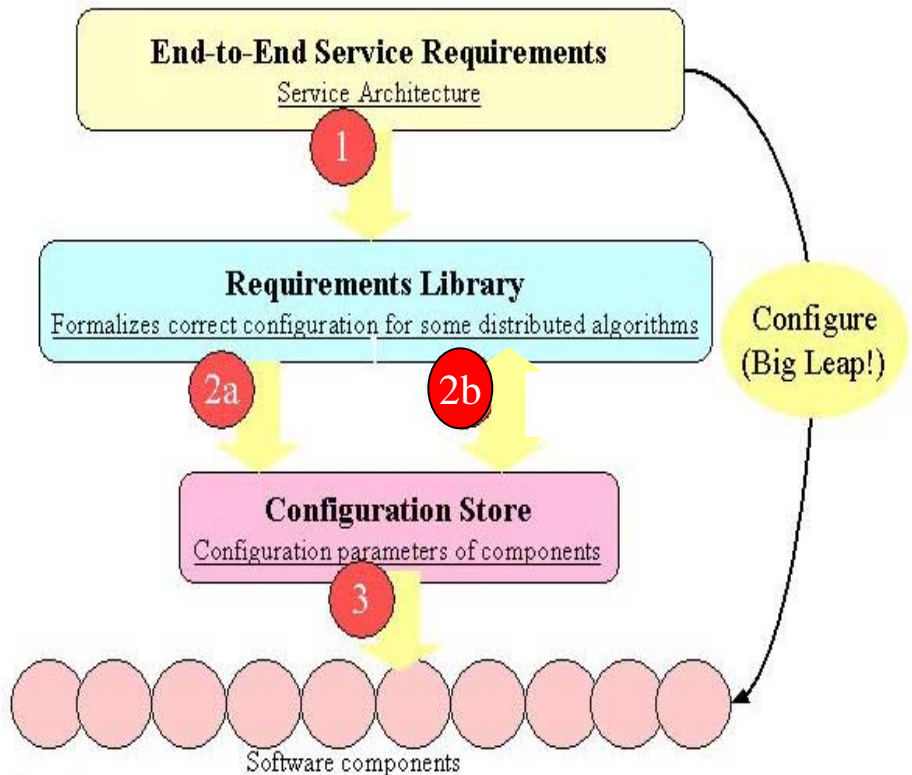


Service Creation using COTS Middleware: Limitation



- Lacks formal method of integration
- Configuration information embedded in object code
- Manual intervention required for changing provisioning information
- Makes it hard to adapt and deploy services dynamically

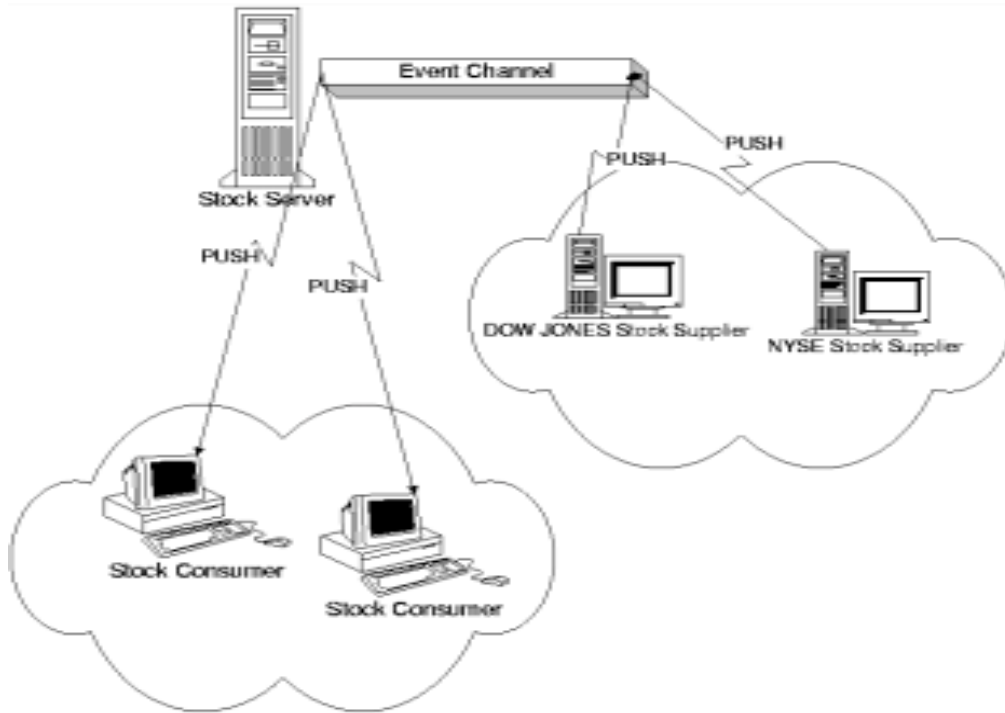
Solution Approach: Service Grammar



1. Decompose
- 2a. Provision (enforce configuration change)
- 2b. Diagnosis (verify configuration)
3. Configuration Generator (generate, deploy configuration)

- Requirements Library defines fundamental building blocks of services
- Compose these to *naturally* specify end-to-end services.
- Provisioning Engine translates services into component configurations
- Diagnosis Engine checks if components configurations are correct
- Distributed Provisioning Engine sets up services in the absence of centralized authority
- Successfully applied to virtual private networks and dynamic coalitions (DARPA)

Setting Up Event Channel Service



■ Problems

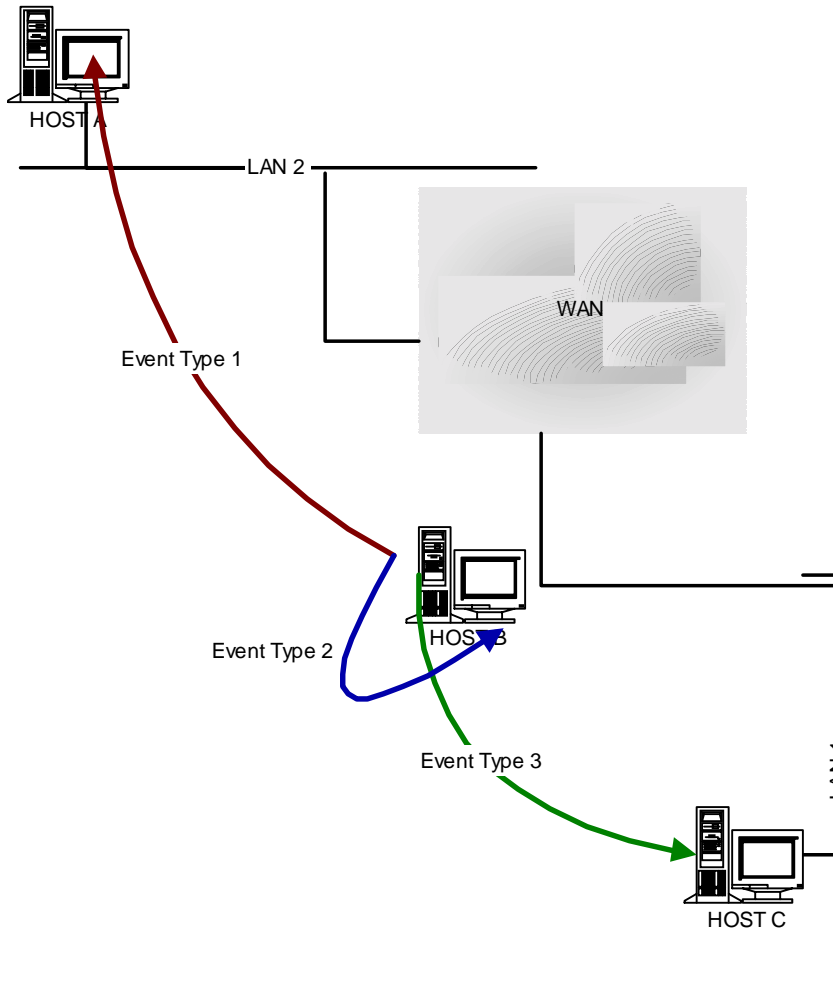
- Attributes have to be computed by application writer and hardcoded into application
- Many errors are possible

- Consumer configuration parameters:
 - Events it subscribes to
 - Event channels it is connected to, and associated consumer administrators and proxy suppliers
 - Naming service
- Supplier's configuration parameters:
 - Events it produces
 - Event channels it is connected to, and associated supplier administrator and proxy consumers
 - Naming service.
- Event channel configuration parameter:
 - Naming service.
- An end-to-end service to set up:
 - Every consumer must receive an subscribed event if a supplier produces it.

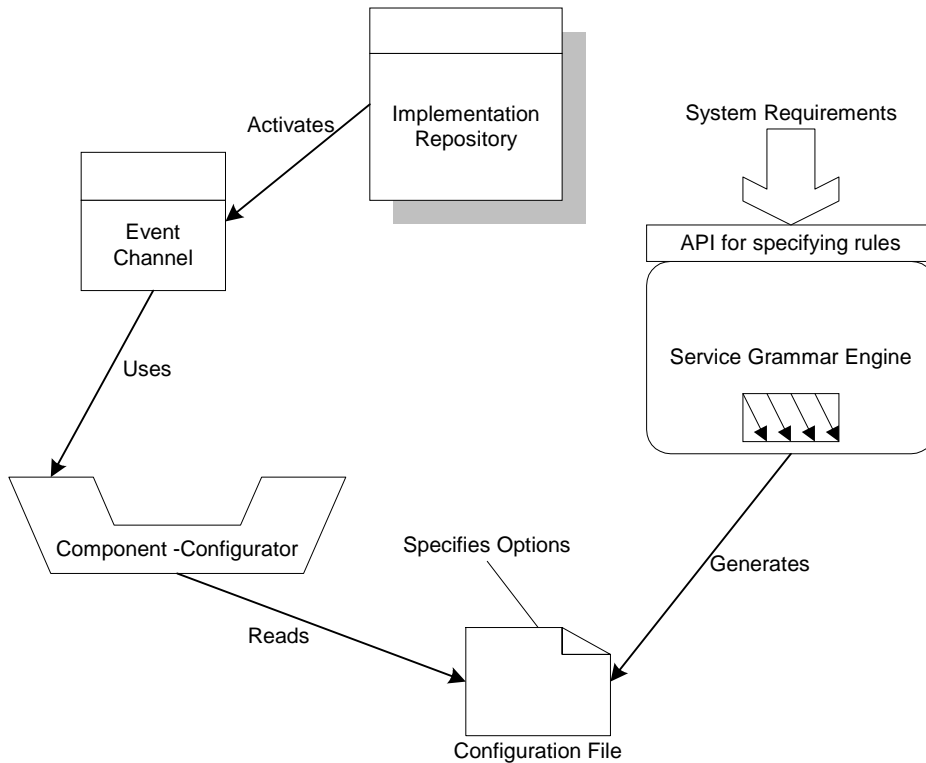
Example

Requirements

- Host B publishes:
 - Event Type 1
 - Event Type 2
 - Event Type 3
- Host A subscribes for:
 - Event Type 1
- Host B subscribes for:
 - Event Type 2
- Host C subscribes for:
 - Event Type 3

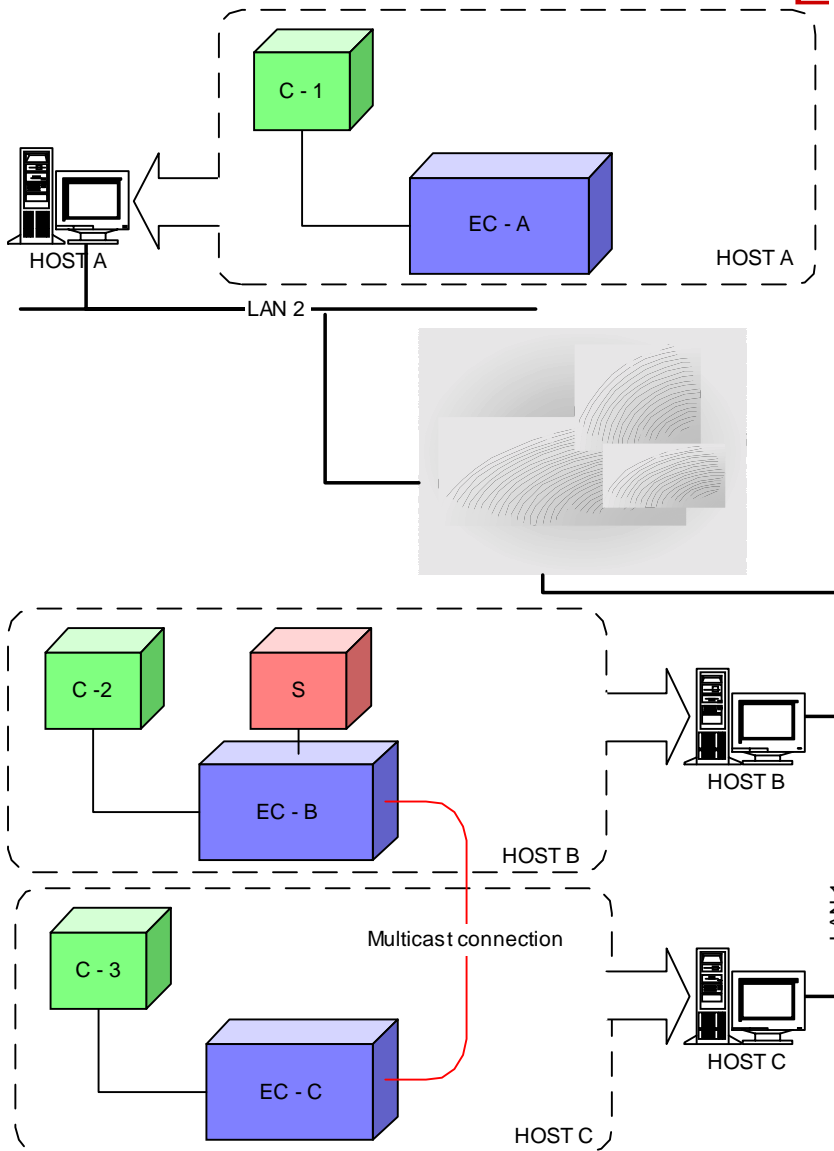


Applying Service Grammar



- Service requirement is specified using Requirements Library for event channels.
- Submitted to Provisioning Engine
- The Engine computes correct configuration of event channels, suppliers and consumers.
- Configuration files are read by the Event Service participants to configure them.
- An Implementation Repository is used to provision the participants on their corresponding hosts

Example



Solution

- Host A runs Event Channel A
- Host B runs Event Channel B
- Host C runs Event Channel C
- Consumer 1 connects to EC A
- Consumer 2 connects to EC B
- Consumer 3 connects to EC C
- Supplier connects to EC B
- EC B connects to EC A as gateway
- EC C and EC B use a multicast connection.

Future Work

- Using Service Grammar approach to compute and verify other requirements for Event Services:
 - Setting up QoS properties for Event Services such as buffer size, concurrency, filters, priority, deadline
 - Verifying event dependencies such as event ordering
 - Configuring Event Channels to communicate via multicast
- Generalizing to middleware services for DRE systems