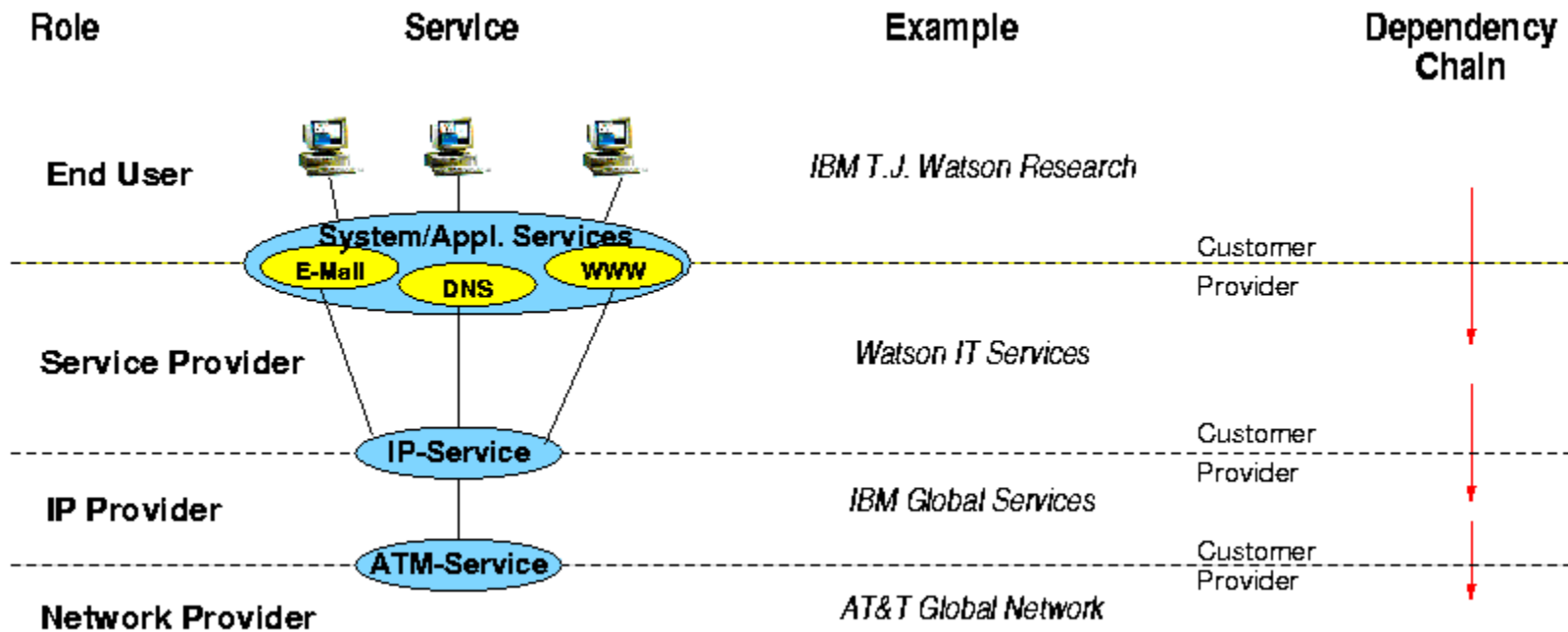




Classification and Computation of Dependencies for Distributed Management

Alexander Keller, Uri Blumenthal, Gautam Kar
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA

Motivation: Managing outsourced Services



- ▼ Dependencies between different Services are layered
- ▼ Explicit Dependencies between Customer/Provider (SLAs)
- ▼ Representation of Dependencies as directed, acyclic graph
- ▼ Client/Server Dependencies usually span multiple systems and domains

Management Issues & Requirements

▼ Root-cause Analysis

- Determine the source of a faulty service
- **Today:** manual test of services by operator, service dependencies not explicitly specified
- **Needed:** “drill-down”, (automated) traversal of layers to find root cause

▼ Impact Analysis

- Determine which services/customers are affected by problem
- **Today:** Planned outages: Proactive warnings, Accidental outages: N/A
- **Needed:** “drill-up”, determine potentially affected services/customers

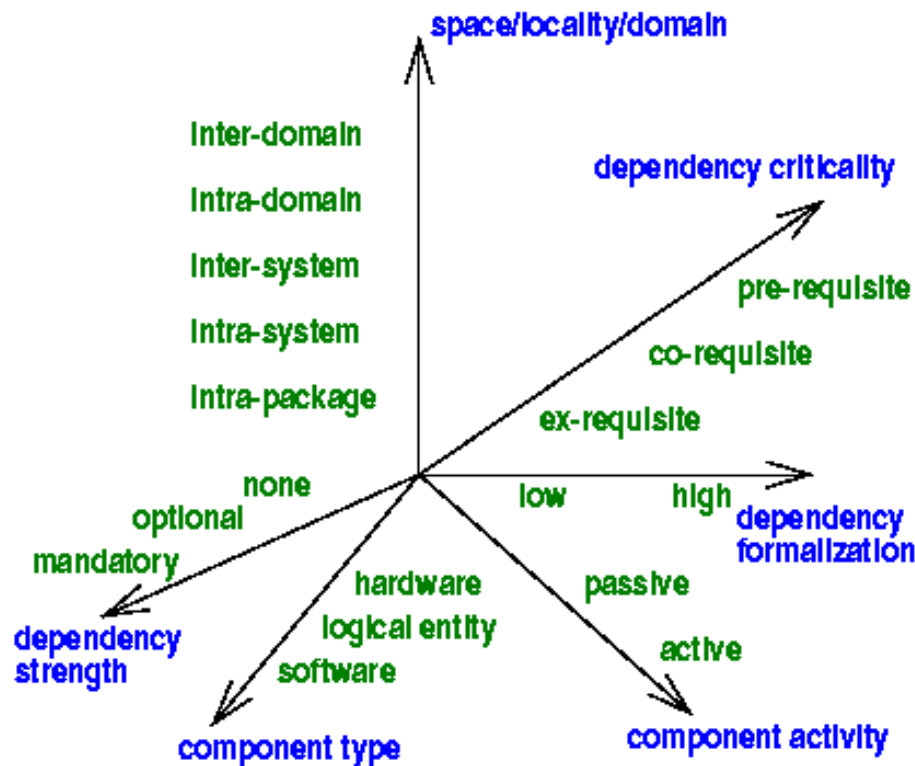
▼ Requirements:

- Make service dependencies explicit and available
- Dependency model must allow upward and downward traversal

Dependencies: State of the Art

- ▼ System Repositories contain Dependency Information:
 - Installation Routine reads SW description
 - adds static SW description in Repository
 - confined to a single system
 - no Application Runtime Information
- ▼ Only two (implicit) kinds of dependencies:
 - *Inter-system* (horizontal) dependencies:
 - ▼ Provide information on client/server relationships
 - ▼ needed for end-to-end problem determination
 - ▼ Example: "Resolver (DNS client) is bound to DNS server"
 - *Intra-system* (vertical) dependencies:
 - ▼ Occur within a single system
 - ▼ Example: "WWW service requires Name service"

Classification of Dependencies



▼ Component-related:

- Space: metric for distance
- Type: nature of component
- Activity: manageability available?

▼ Dependency-related:

- Strength: degree of dependency
- Formalization: cost of acquisition
- Criticality: compatibility measure

▼ Lifecycle-related:

- Application states:
 - ▼ Deployable, Installable, Executable, Running
- Pre-conditions, Post-conditions
- Impacts previous categories

Making use of System Repositories

▼ Main Idea:

- Extract information on dependencies between installed Software
- Infer runtime behavior of Services from existing data

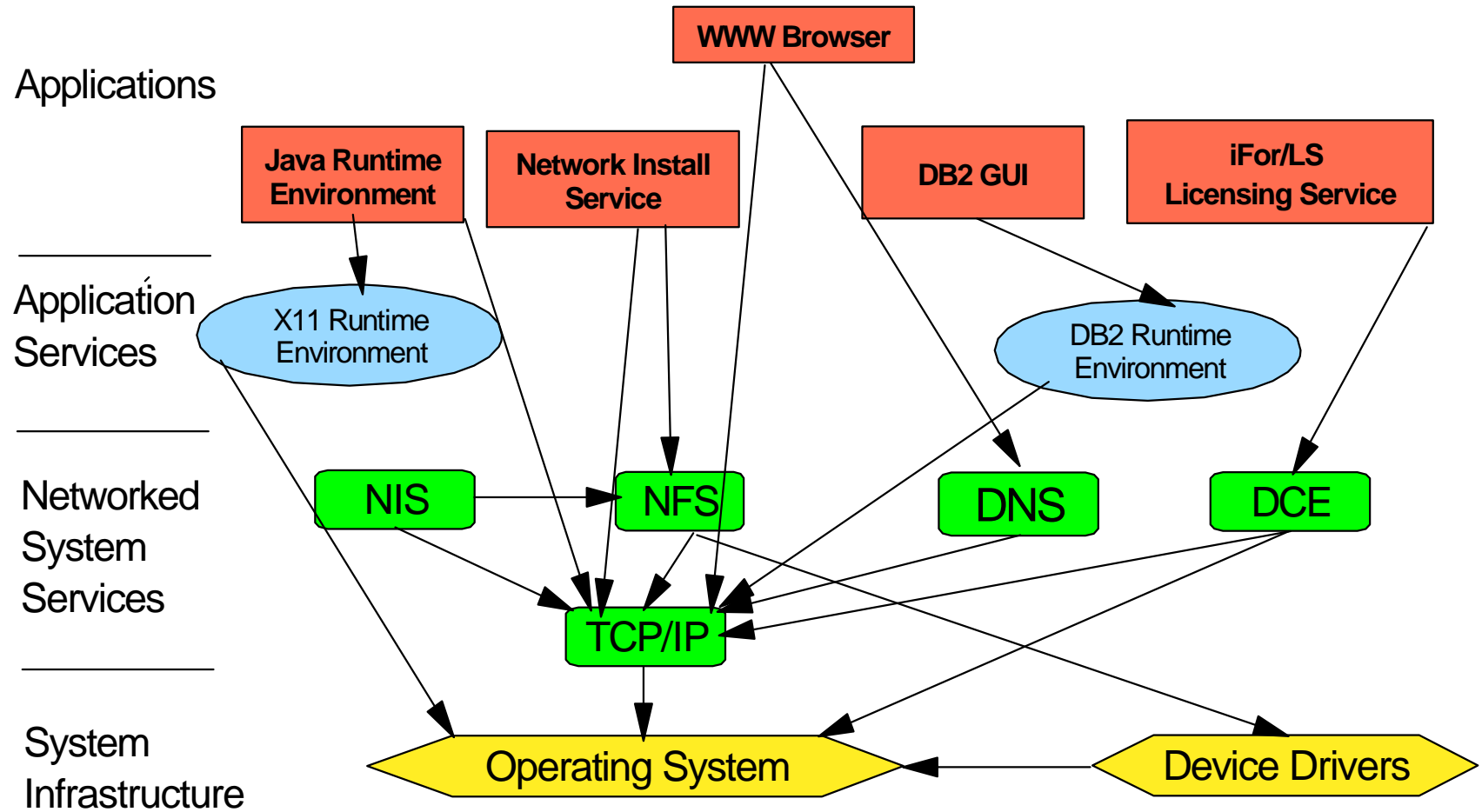
▼ Repositories already contain useful information:

- SW package names, install status, version #, fix#, description...
- dependencies between the layers within a system
- hints on client/server bindings between different systems
- compatibility information w.r.t. versions & levels
- hierarchical naming conventions, notion of containment

▼ Example from AIX Object Data Manager:

- "Prerequisites" field of bos.net.tcp.server:
- "bos.rte 4.2.0.0, bos.net.tcp.client 4.1.0.0"

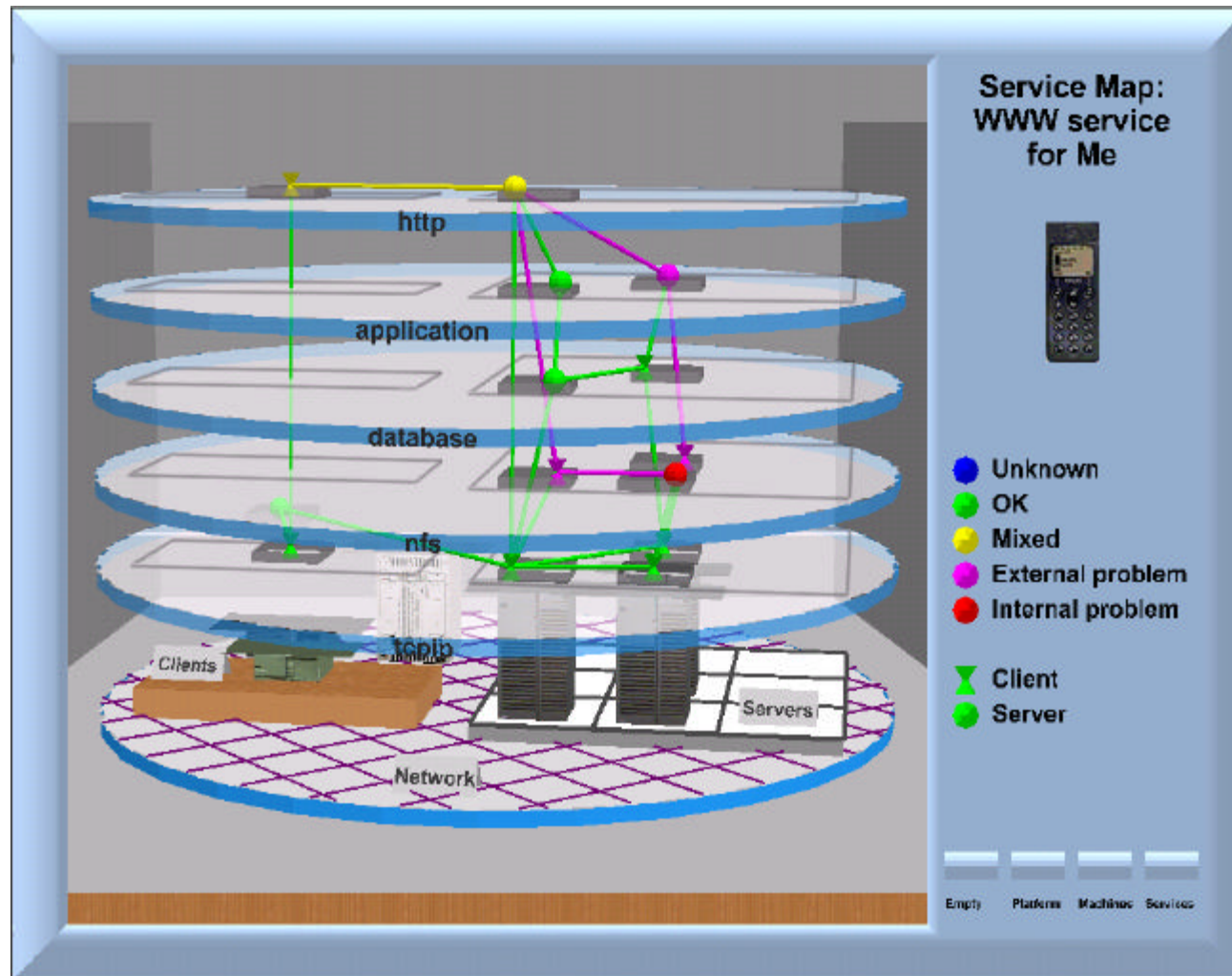
Functional Dependency Model (Extract)



Implementation Issues

- ▼ **Dependency Model derived from Common Information Model**
 - Based on CIM Core, System and Application Schemas
 - Addresses packages, applications, services, subcomponents and their relationships (containment, dependencies, potential bindings)
- ▼ **Prototype components implemented in Java**
 - Communication through Java Remote Method Invocation
 - Existing Repository interfaces shielded by Java Native Interface
- ▼ **Enhanced Repository for Operating Systems (AIX, Linux)**
 - Single place for determining dependency information
 - Includes missing information:
 - ▼ links processes to services
 - ▼ Looks up service / port mappings
 - ▼ Contains links to further configuration information

Prototype GUI: Design Study



Conclusions and Outlook

▼ Results:

- Automated determination of static dependencies
- Obtained from operating system repositories
- Identifies functional dependencies between application services
- Small extensions to determine potential dependencies
- Extend CIM Application Schema
- Distinguishing between instances is hard, identification of effective bindings missing yet

▼ Current work:

- Cross-Domain (End-to-End) Dependencies
- Address runtime aspects of applications
- Model state transitions according to application lifecycle
- Integration with available CIM Object Manager