Defining a Technical Basis for Comparing and Contrasting Emerging Dynamic Discovery Protocols

Christopher Dabrowski  
Software Diagnostics & Conformance Testing Division  
cdabrowski@nist.gov

Kevin Mills  
Advanced Networked Technologies Division  
kmills@nist.gov

PC2001 Conference  
May 2, 2001
Modeling Function, Structure, and Behavior

Objectives

1. Provide increased understanding of the competing dynamic discovery services emerging in industry
2. Develop metrics for comparative analysis of different approaches to dynamic discovery and assuring quality and correctness of discovery protocols
3. Assess suitability of architecture description languages to model and analyze emerging dynamic discovery protocols

Technical Approach

- Develop ADL models from selected specifications for service discovery protocols and develop a suite of scenarios and topologies with which to exercise the ADL models
- Propose a set of consistency conditions & constraints that dynamic discovery protocols should satisfy
- Propose a set of metrics, based on partially ordered sets, with which to compare and contrast discovery protocols
- Analyze ADL models to assess consistency condition satisfaction, and to compare and contrast protocols

Status as of May 1, 2001

- Developed a generic UML model encompassing the structure and function of Jini, UPnP, SLP, Bluetooth, and HAVi
- Projected specific UML models for Jini, UPnP, and SLP
- Completed a Rapide Model of Jini structure, function, and behavior
- Drafted and implemented a scenario language to drive the Rapide Jini Model.
- Developed a set of consistency conditions and constraints for Jini behavioral model; currently being tested using scenarios.
- Discovered significant architectural issue in interaction between Jini directed discovery and multicast discovery

Products

- Rapide specifications of Jini, Universal Plug and Play (UPnP), and Service Location Protocol (SLP)
- Scenarios and topologies for evaluating discovery protocols
- Suggested consistency properties for service discovery protocols
- Suggested metrics, based on partially ordered sets (POSETs), for comparing and contrasting discovery protocols
- Paper identifying inconsistencies and ambiguities in service discovery protocols and describing how they were found
- Paper proposing consistency conditions for service discovery protocols, and evaluating how Jini, UPnP, and SLP fare
- Paper comparing and contrasting Jini, UPnP, and SLP at the level of POSET metrics
What is a dynamic discovery protocol?

Dynamic discovery protocols enable dynamic elements in a network (including software clients and services, as well as devices):
(1) to discover each other without pre-arrangement,
(2) to express opportunities for collaboration, and
(3) to compose themselves into larger collections that cooperate to meet an application need.

What about robustness in the face of change?

Discovery protocols contain logic intended to provide resilience in the face of process, node, and link failures of both a temporary and permanent nature.
Various Protocols for Dynamic Discovery are Emerging in the Commercial Sector
Why are various dynamic discovery protocols emerging?

- Some industry groups approach the problem from a vertically integrated perspective, coupled with a narrow application focus but targeting a different application domain. (e.g., HAVi, Salutation Consortium, and Bluetooth Service Discovery)

- Sun has designed Jini as a general service-discovery mechanism atop Java, which provides a base of portable software technology.

- Some suspect that the Jini approach will prove too inefficient for use in consumer appliances and in other low-cost, low-performance computing platforms; thus, some propose a different set of protocols.
Our Motivation?

To provide industry with metrics to compare and contrast emerging dynamic discovery protocols and to strengthen the quality and correctness of those protocols.

Our General Approach?

1) Use Architectural Description Languages (ADLs) and associated tools to analyze Discovery Protocol specifications assessing consistency and completeness w.r.t. conditions of dynamic change.

2) Compare and contrast emerging commercial service discovery technologies with regard to function, structure, behavior, performance and scalability in the face of dynamic change.
Particulars of Our Approach

- Define a Generic UML Model that Encompasses Jini, UPnP, SLP, HAVi, and Bluetooth Service Discovery and that provides a common terminology for discussing discovery protocols, and then derive Specific UML Models for Jini, UPnP, & SLP (expressed in the common terminology)

- From the UML models and the specifications, encode executable Rapide models of the structure and behavior of Jini, UPnP, & SLP

- Define consistency conditions and constraints that should be satisfied by discovery protocols in general and by specific discovery protocols, and define some behavioral metrics

- Define a scenario language and scenarios to drive the executable models, and then exercise the models while evaluating satisfaction of consistency conditions and constraints and assessing the behavior exhibited by the executable models
Generic UML Structural Model of Service Discovery Protocols

**SERVICE MANAGER**
- discover Network Context()
- <<not shr>> Cache Manager Discovery()
- <<OPT>> Service Manager()
- <<not shr>> start Service Parameter Matching Task()

**SERVICE PROVIDER**
- SERVICE DESCRIPTION
  - identify
  - Type
  - API
  - Attributes
- SERVICE Repository

**SERVICE USER**
- SERVICE DISCOVERY
- <<not shr>> start Service Parameter Matching Task()

**SERVICE CACHE MANAGER**
- discover Network Context()
- <<not shr>> activate Manager Discovery()
- Service Manager()
- start Aging Task()
- Service Cache Manager()

**LOCAL CACHE MANAGER**
- Start Aging Task()

**SERVICE USER**
- discover Network Context()
- Service DISCOVERY
- <<not shr>> start Renewal Task()
- Service User()

**NOTIFICATION REQUEST**
- Notification Request (from Data View)

**SERVICE CACHE**
- Notification Cache
- Service Cache
- Service Repository
- Service Parameter Change Notification
- Parameter Notification Request

**SERVICE USER**
- queries information from
- invokes operations

**SERVICE PROVIDER**
- manages
- contains

**SERVICE MANAGER**
- contains
- manages

**SERVICE USER**
- invokes operations

**SERVICE PROVIDER**
- discover Network Context
- Service DISCOVERY
- <<not shr>> start Renewal Task()

**SERVICE USER**
- requests
- availability

**SERVICE MANAGER**
- activate Manager Discovery
- Service Provider()
- start Matching Task()
- Service Manager()

**SERVICE USER**
- requests
- service

**SERVICE USER**
- requests
- service

**SERVICE USER**
- requests
- service

**SERVICE MANAGER**
- activate Manager Discovery
- Service Provider()
- start Matching Task()
Projected UML Structural Model of Jini

SERVICE MANAGER
- Discover Network Context()
- Cache Manager Discovery()
- Announce Service Processing()
- Start Renewal Task()
- Service Manager()

SERVICE PROVIDER
- SERVICE DESCRIPTION
  - Identify
  - Type
  - API
  - GUI
  - Attributes

SERVICE USER
- discover Network Context()
- Service Discovery()
- Start Renewal Task()
- Service User()

SERVICE CACHE MANAGER
- discover Network Context()
- Service Manager Discovery()
- Service Manager()
- Service Cache Manager()
- Service Cache()
- Notification Cache

<<repository entry>>
Service Repository
- Contains
- manages
- Aggregates

<<repository entry>>
SERVICE DESCRIPTION
- from Data View
- query information from

<<repository entry>>
Notification Request
- from Data View
Architecture Description Languages and Tools

Allow us to model the essential complexity of discovery protocols, while ignoring the incidental complexity

Jini is documented in a 385 page specification; however, the static nature captures only the normative complexity because most of the essential complexity arises through interactions among distributed, independently acting Jini components.

Incidental complexity represented by the code: for example consider Core Jini – an 832 page commentary on the massive amount of Java code that comprises Jini, which also depends on complex underlying code for Remote Method Invocation, Distributed Events, Object Serialization, TCP/IP, UDP, HTTP, and Multicast Protocol Implementation.
Architectural Description Languages

• Provide effective abstractions for representing and analyze software architectures (components, connections, behavior, constraints, etc.)
  – Using Rapide (Stanford) because of POSET paradigm & constraint language

• ADLs provide a framework and context
  – to more easily pinpoint where inconsistencies and ambiguities may exist within software implementing specifications & to understand how they arise
  – to define metrics that yield qualitative and quantitative measures of dynamic component-based software

• ADLs provide basis to compare and contrast dynamic discovery protocols (Jini, UP&P, SLP)
Rapide, an Architecture Description Language and Tools
Developed for DARPA by Stanford

MODELING ESSENTIAL COMPLEXITY

Model Specification in Rapide

```
... *******************************************************
** 3.3 DIRECTED DISCOVERY CLIENT INTERFACE **
... *******************************************************
-- This is used by all JINI entities in directed
-- discovery mode. It is part of the SCM_Discovery
-- Module. Sends Unicast messages to SCMs on list of
-- SCMs to be discovered until all SCMs are found.
-- Receives updates from SCM DB of discovered SCMs and
-- removes SCMs accordingly.
-- NOTE: Failure and recovery behavior are not
-- yet defined and need review.

TYPE Directed_Discovery_Client
  (SourceID : IP_Address; InSCMsToDiscover : SCMList; StartOption : DD_Code;
   InRequestInterval : TimeUnit; InMaxNumTries : integer; InPV : ProtocolVersion)
IS INTERFACE
SERVICE DDC_SEND_DIR : DIRECTED_2_STEP_PROTOCOL;
SERVICE DISC_MODES : dual SCM_DISCOVERY_MODES;
SERVICE DD_SCM_Update : DD_SCM_Update;
SERVICE SCM_Update : SCM_Update;
SERVICE DB_Update : dual DB_Update;
SERVICE NODE_FAILURES : NODE_FAILURES;
ACTION
  IN Send_Requests(),
  BeginDirectedDiscovery();
  BEHAVIOR
    action animation_Iam (name: string);
  MySourceID : VAR IP_Address;
  PV : VAR ProtocolVersion;

... *******************************************************
```

Execute with Raptor Engine

Analyze Generated POSETs

Assess Consistency
Condition Satisfaction & Constraint Violations
Define Executable JINI Architectural Model in Rapide

Network Topological Entities
- Network Node

JINI Entities
- Service Manager
- Service Cache Manager
- Service User

Entity Major Functions
- Service Repository
- SCM Discovery
- SCM Beacon & Response
- SCM Matching Cache
- Notification Repository

Entity Functions
- Lazy Discovery
- Directed Discovery
- Aggressive Discovery
- Lazy Discovery

Legend
- Agency Type of Part of
Deploy Instances of Jini Entities and Communication Channels in a Topology
Drive Model Topology with Scenarios

> StartTime  {NodeFail || NodeRecover}  NodeID DelayTime.
> StartTime  {LinkFail || LinkRestore}  NodeID DelayTime FromNode ToNode.
> StartTime  {MProbeFail || MProbeRestore}  NodeID DelayTime FromNode ToNode.
> StartTime  {GroupJoin || GroupLeave}  NodeID DelayTime.
> StartTime  {AddSCM || DeleteSCM}  NodeID DelayTime.
> StartTime  {AddService ChangeService} NodeID DelayTime ServiceTemplate ServiceAPI ServiceGUI LeaseTime DurationTime.
> StartTime  DeleteService NodeID DelayTime ServiceID.
> StartTime  FindService NodeID DelayTime SMNodeID .
> StartTime  AddNotificationRequest NodeID DelayTime NotificationID ServiceTemplate Transitions LeaseTime DurationTime SCMID.
> StartTime  DeleteNotificationRequest NodeID DelayTime NotificationID SCMID.
Analyze Consistency Condition Satisfaction in Real-Time

Sample Consistency Conditions*

For All (SM, SD, SCM): (SM, SD) IsElementOf SCM registered-services implies SCM IsElementOf SM discovered-SCMs

For All (SU, NR, SCM): (SU, NR) IsElementOf SCM registered-notifications implies SCM IsElementOf SU discovered-SCMs

*Assuming absence of network failure and normal delays due to updates

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager
- SU is Service User
- NR is Notification Request
- Registered-services is a set of (SM, SD) pairs
- Registered-notifications is a set of (SU, NR) pairs
- Discovered-SCMs is a set of SCM
Analyze POSETs Off-Line to Compare and Contrast Behaviors Given a Congruent Topology and Scenario

Metrics Based on Numbers of Messages
• Message volume?
• Message intensity?

Metrics Based on Complexity
• Degree of dependency among messages?
• Rate of consistency & constraint violations?
• Rate of exceptions?

Metrics Based on Time
• Service latency?
• Service throughput?
• Recovery latency?

Metrics Based on Change
• Derivative of the message intensity?
• Derivative of the service throughput?
• Derivative of the service latency?

POSET analyses provide basis for defining metrics that provide quantitative measures of properties of a system
Where do we stand now?

- Developed an initial UML model of a generic service discovery protocol with specific projections for Jini, UPnP, and SLP
- Developed and exercised an executable Rapide model of the Jini, including some consistency conditions, constraints, behavioral metrics
- Providing developers of Jini (and of other protocols) with results of analysis
- Working on a paper to describe our goals, approach, and interim results, and to provide recommendations for improving and using ADLs.

What’s to come?

- Develop, exercise, and analyze executable specifications for Universal Plug-and-Play (UPnP) and Service Location Protocol (SLP)
- Provide a technical comparison among Jini, UPnP, and SLP
If you want to see a demo, then please let me know? cdabrowski@nist.gov