Auto-ID Center Software Framework

Agenda

- Objective of the meeting
- Overview of the Object Name Service (ONS)
- Demonstration of the ONS
- Break
- Overview of the Savant
- Demonstration of the Savant
- Questions?

Objective of the Meeting

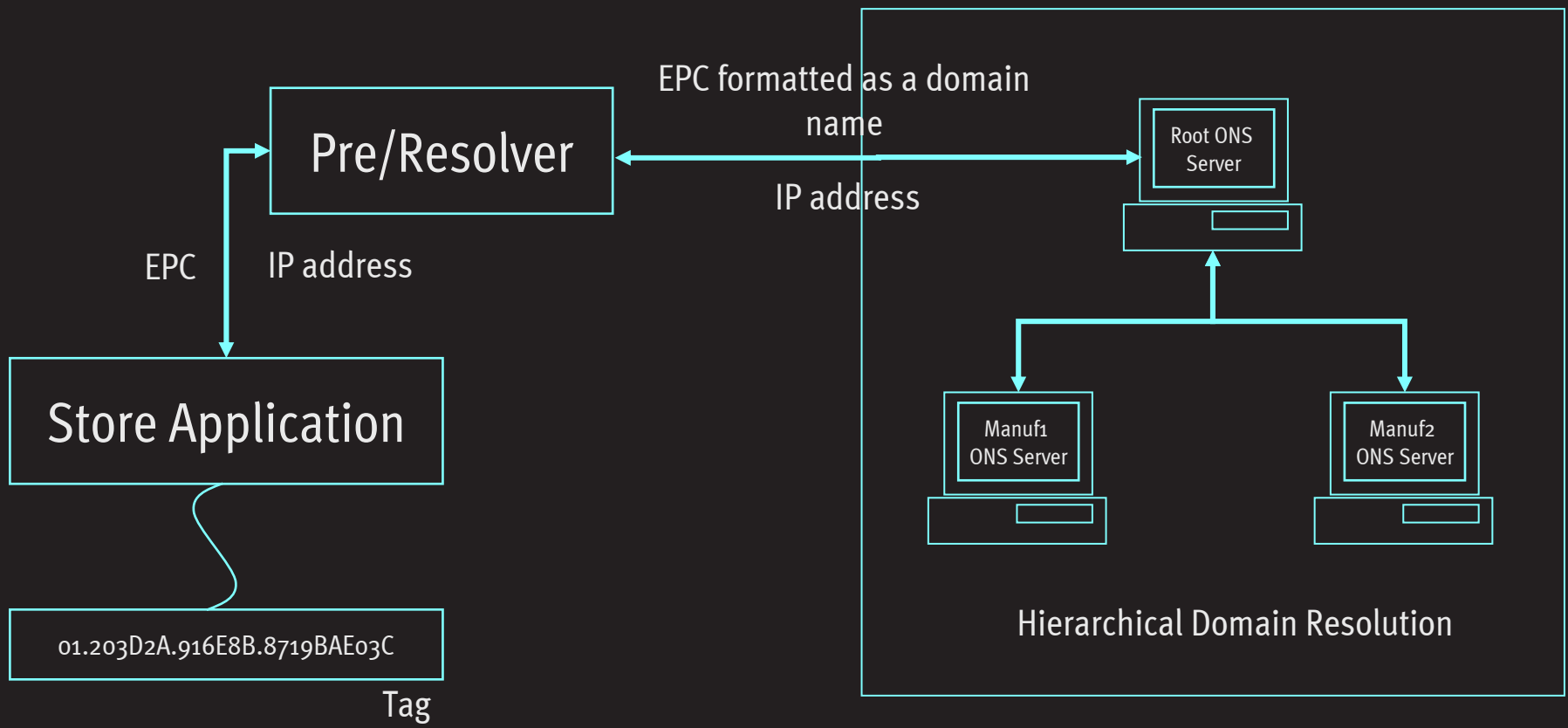
- Present an overview of Auto-ID center's software framework
- Present the design choices and get feedback for the next version

Overview of the Object Name Service (ONS)

ONS Server

- The Object Name Service (ONS) provides a framework for locating networked services (PML servers) for objects tagged with EPCs.
- ONS is built over the existing DNS framework
- Given an EPC the ONS Framework will either:
  - Return the IP address of the PML server, at the manufacturer, holding additional information about the EPC
  - Return the IP address of an internal server to which the information about the EPC can be written to

How does ONS Server Work – High Level



Example

1. EPC (96-bit)

   01.203D2A.916E8B.8719BAE03C

   Version Manufacturer Product Serial

2. EPC converted to a domain name by the pre/resolver

   8719BAE03C.916E8B.203D2A.01.epc.objid.net

   Serial . Product . Manufacturer . Version . Root domain

3. Domain name resolution by ONS Server

   203D2A.01.epc.objid.net [Object Id Root Server]

   916E8B.203D2A.01.epc.objid.net [Manufacturer server]

   8719BAE03C.916E8B.203D2A.01.epc.objid.net [Product server]

ONS Server Components

- Pre-resolver: The ONS pre-resolver computes the EPC domain name associated with an EPC.
- Resolver: A standard DNS lookup on the EPC domain name will list the PML servers associated with that EPC. [OS, BIND lwres, GNU adns]
- DNS server: A DNS server that holds mapping information between EPC domain names and IP addresses of associated PML servers. [BIND]
- Server configuration tool: An XML processor that generates BIND configuration files, given an ONS specification file
- Specification management tool: An XML processor that updates ONS specification files given ONS update files.
- Content Server: Stores the mapping information in a database and serves the specification XML files to one or more ONS severs

Pre-Resolver

- Translates EPC number to EPC domain name
  - Simple Translation
  - Translation using formatting strings

Pre-Resolver – Simple Translation

Input

01.203D2A.916E8B.8719BAE03C

Output:

8719BAE03C.916E8B.203D2A.01.epc.objid.net

This assumes that the pre-resolver knows the EPC partitioning

Data Flow – Simple Translation

Pre-Resolver – Translation Using Formatting Strings

1. EPC (96-bit)

   01.203D2A.916E8B.8719BAE03C

   Version . Manufacturer . Product . Serial

2. The pre-resolver recognizes the version to be 01 The pre-resolver queries the ONS server for the info record

   info.01.epc.objid.net

3. The ONS server returns a TXT record with the format string

   4444.4444.4.4.4444.4.4.44444.44

4. EPC converted to a domain name by the pre/resolver

   E03C.19BA.7.8.6E8B.1.9.203D2A.01.epc.objid.net

Pre-Resolver – Translation Using Partial-Formatting Strings

- If the ONS Server specifies partial formatting string then the pre-resolver will ignore the least significant bits

For example: Lets consider

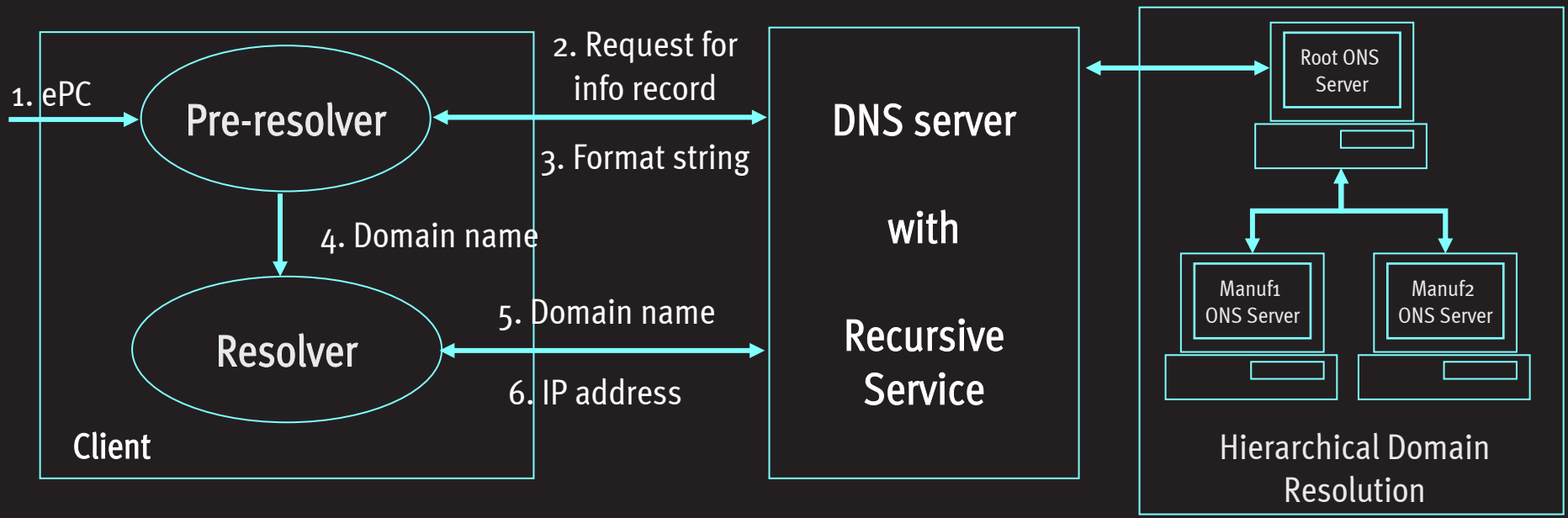
EPC (64 bits):

01101010011110010110100111001011110100011010110001101101011001 00

Formatting String (40 bits): 4.4.4444.1.1.1.3.3.3.013

Since the EPC has 64 bits, the least significant 24 bits are truncated resulting in: 01101010011110010110100111001011110100 01.

Data Flow – Translation Using Formatting Strings



AUTO-ID CENTER

Pre-Resolver – How to use it

- The pre-resolver is built on a stub-resolver. The DNS server performing lookups for the pre-resolver should provide recursive lookup service.
- Command Line Interface

```
onslookup <bit-string> <domain-suffix> <name-server>
```

Pre-Resolver – C Interface

```
int initialize_ons_resolver(ons_resolver_state *state,
                            const char *epc_domain_suffix,
                            const char *resolver_conf,
                            int iterative_lookup);

int get_epc_domain(ons_resolver_state state,
                   char *result_domain_name,
                   int num_epc_bits,
                   const char *epc);


int get_epc_domain_2(char *result_domain_name,
                     int num_epc_bits,
                     const char *epc,
                     const char *epc_domain_suffix,
                     int flags);
```

Pre-Resolver – Java Interface

- `public static String getEpcDomain(int numBits, String epc, String epcDomainSuffix, boolean iterative, String nameServer);`

Demo of Pre-Resolver

- Converting EPC:00000000100000100000012345 to a domain name using formatting strings
- Sample code that performs this lookup

```
public static String getEpcDomain(String epc,
String domainSuffix) throws ResolverException {
    return Resolver.getEpcDomain(96, epc,
                domainSuffix, false, null);
}
```

ONS Configuration/Maintenance

ONS Specification File

- An ONS specification consists of the following components:

  High-level configuration information: Trusted subnet, external and internal server definitions, administration keys, hint and zone files.

  Content information: Authoritative zones, secondary (slave) zones, host and name server entries.

- The ONS specification file is an XML file with syntax defined in `ons specification.dtd`

- Sample ONS specification file, ONS content file

Server Configuration Tool

- The server configuration tool (`onsconfig`) reads an ONS specification file to generate the required DNS configuration and zone files.
- The server configuration tool is written in XT and Java
- Command line interface

```
onsconfig <type:internal/external> <input config
file> --bothservers --zonesonly
```

Configuration Management Tool

- The ONS content information can be managed using the configuration management tool (`onsupdate`).
- Sample ONS configuration update file
- Command line interface

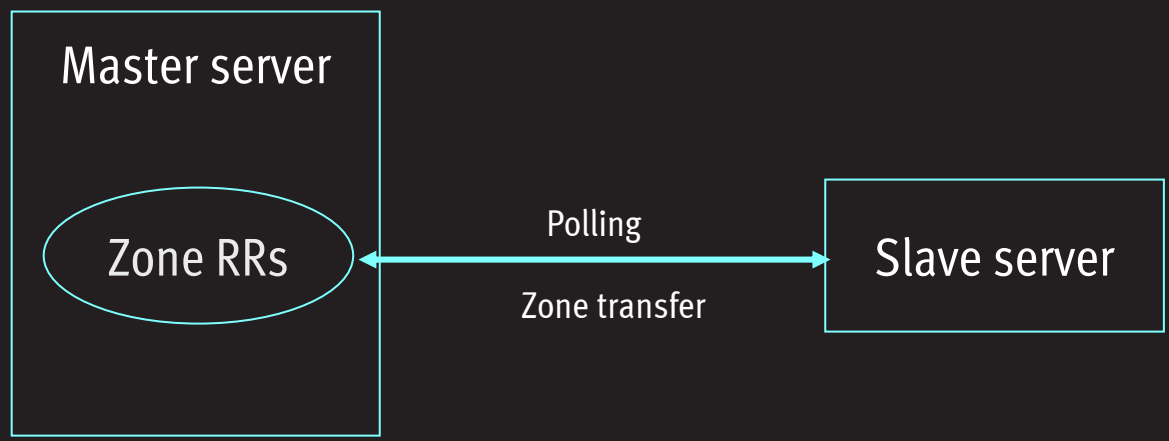
```
onsupdate <input file> <update file> <output file>
<error file>
```

Content Server

- Content Server stores the mapping information in a database and serves the specification XML files to one or more ONS servers.
- Content Server is implemented as a webserver serving ONS specification files over HTTP
- Master ONS content file served by the Content Server
- Slave ONS content file served by the Content Server

Redundancy – Master/Slave Configuration

- To improve the reliability of ONS servers, one or more slave servers can be configured. Slave servers refresh the authoritative information using *zone transfers.* Alternately, the master servers can *notify* the slaves regarding the updates made to the zone



AUTO-ID CENTER

Caching

- The DNS server offering recursive service will perform caching as follows:
  1. While performing a lookup, the server will check its cache first for the closest match, and then continue the search.
  2. The cache holds all RRs that are still valid according to the associated TTL (time to live) parameter. This is called *positive caching*.
  3. The cache also holds the queries for which no RR was found. Negative entries are valid for the TTL specified for the zone. This is called *negative caching*.
- Negative TTL determines the time taken for a new EPC range entry to take effect.
- Positive TTL determines the time for which a PML server should be active after it is removed from the ONS framework.
- Positive and Negative TTL

Redundancy & Caching - Demo

- Caching

  Positive caching: EPC:00000000100000100000012345

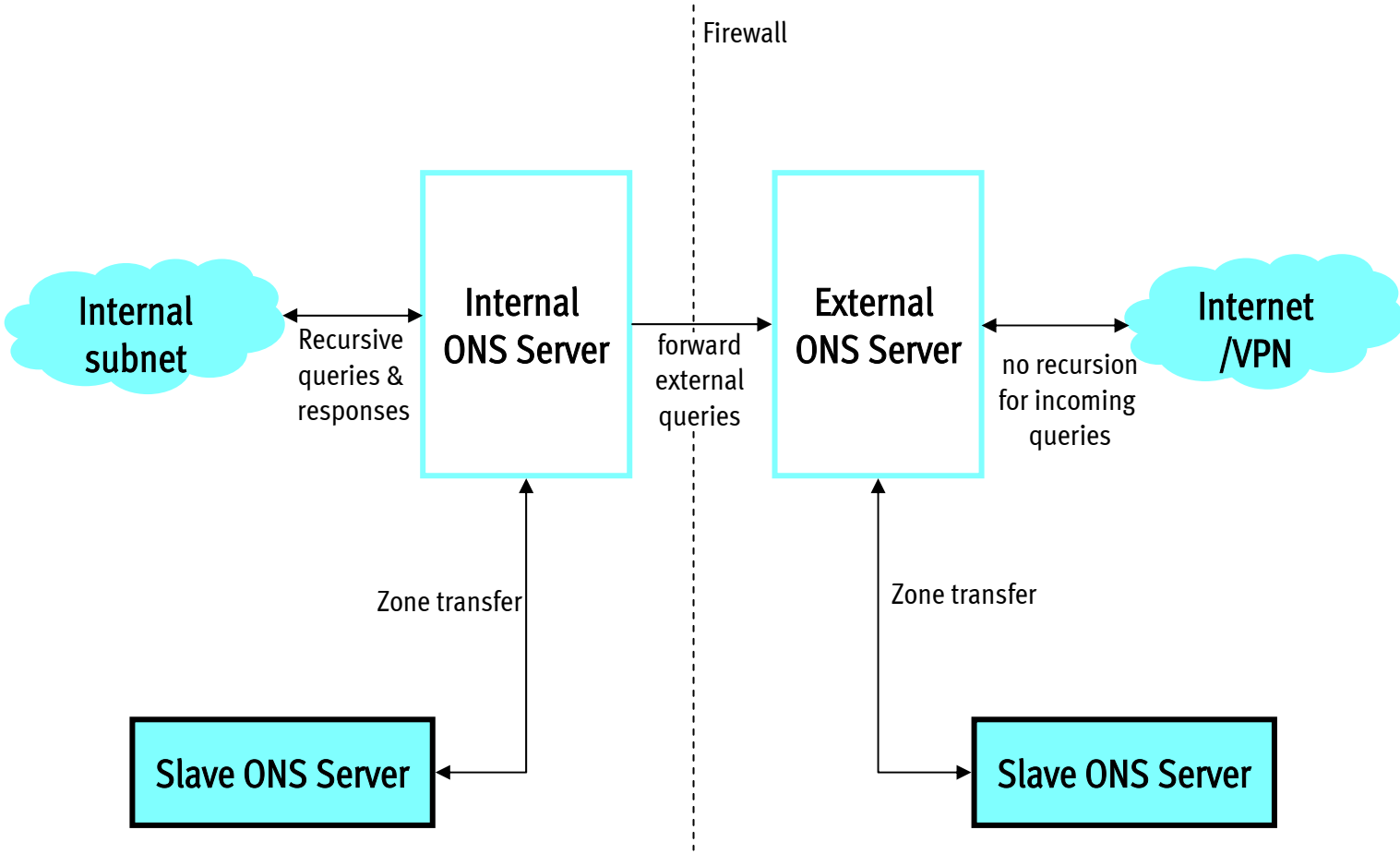
  Negative caching: EPC:00000001100000100000012345

Redundancy

  Disconnect the master server, and perform lookup on a EPC

  EPC:00000000200000200000012345

Security – Trusted Subnets



Technology

- The ONS resolver uses the GNU adns & dnsjava libraries to perform DNS lookups
- ONS servers are implemented by appropriately configuring BIND (Berkeley Internet Name Domain) servers.
- The configuration tool is written using XT (an implementation of XSLT with some extensions). An XSLT file transforms the given XML specification files to BIND configuration files. XT extension functions allow the invocation of Java methods during the XSL transform.

Performance Statistics

- Hardware & Software Configuration:

  DELL PowerEdge Server 2500, with 1133MHz Intel Pentium III processor, 512KB cache, and 512MB RAM, running BIND 9.1.3

- Test Scenario

  Local machine or machine accessible through 100 Mbps Ethernet LAN

- Performance

  0.27 ms/ONS Lookup (~3,700 ONS Lookups/sec)

AUTO-ID CENTER

Future Work

- Look into the possibility of using SRV & TXT records for connection provisioning
  - Port number of the PML server
  - Services exposed by the PML server
  - Connection mechanism
  - Security & Encryption schemes

Questions?

Overview of Savant

SAVANT

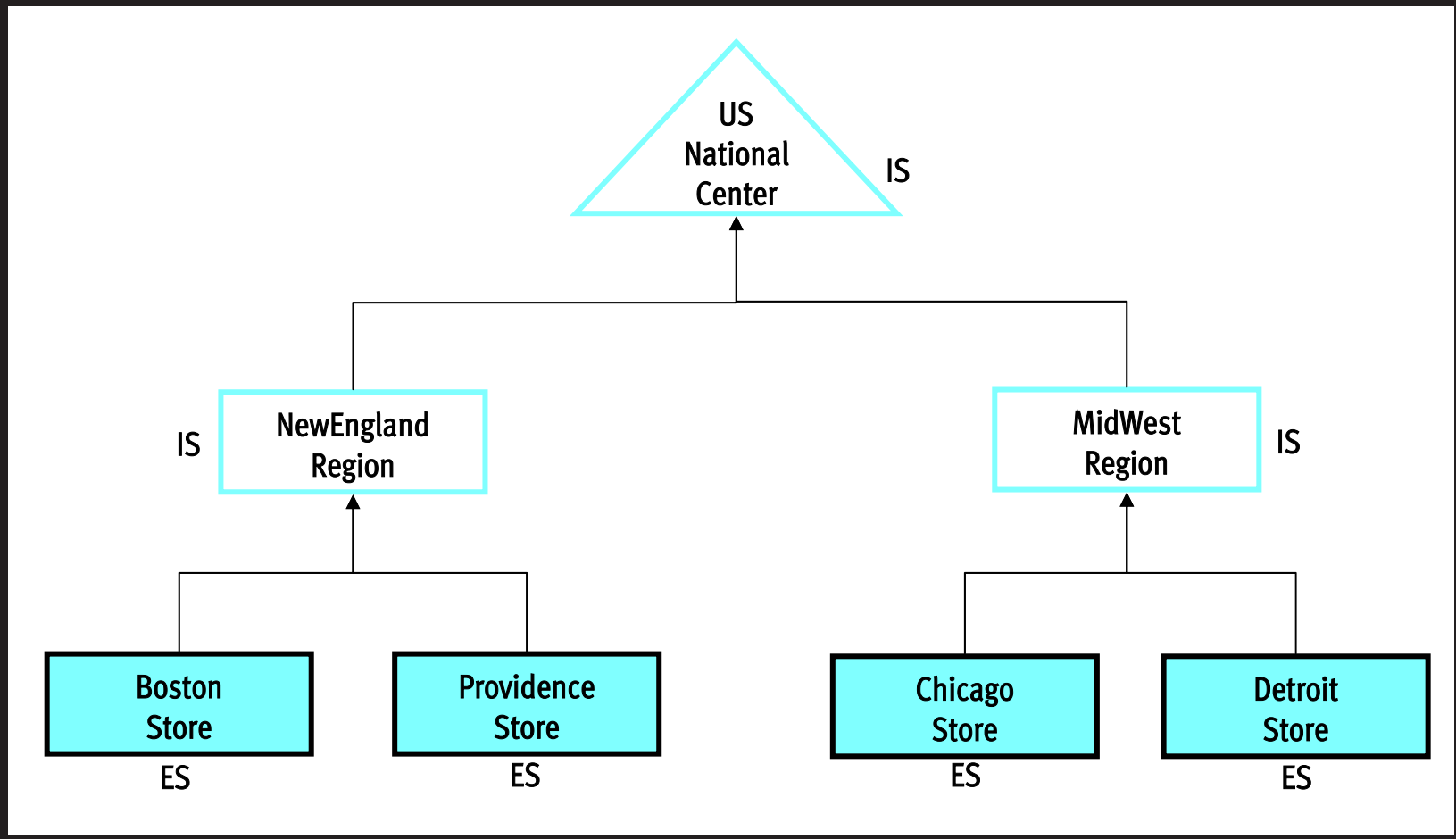
- The Savant is a data router that performs operations such as data capturing, data monitoring, and data transmission

AUTO-ID CENTER

Requirements

- System that can capture high volume of EPC events from multiple readers
- Platform independent implementation
- Self-replicating system that can be installed at the store level or at a regional data center
- Distributed data management and monitoring
- Interface to share EPC data with other applications

Sample Savant Network



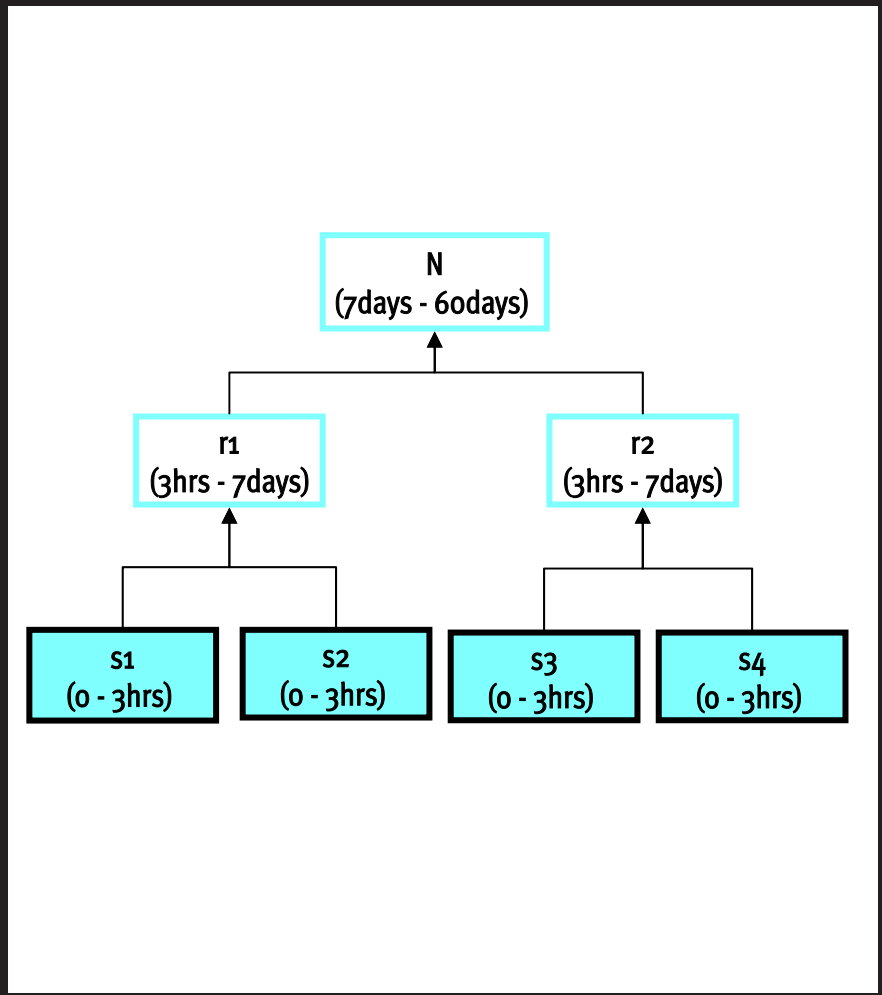
Edge Savants

- Edge Savants (ESs) are the leaf nodes in the savant network. Typically, these savants reside at the stores, warehouses and manufacturing plants
- ESs are connected to the RF readers and capture, monitor, log real-time EPC data from the readers
- For each read the savant maintains:
  - Tag EPC
  - Reader EPC
  - Timestamp
  - Non-EPC information such as temperature, geographical information etc

Internal Savants

- Internal Savants (ISs) are internal nodes of the savant network. They collect EPC data from their descendents . These descendents could be ES or IS.
- Internal Savants are typically located at the regional or national data centers of an enterprise
- In addition to maintaining tag EPC, reader EPC and timestamp the Internal Savant also maintains the 'Source Node' of the data.

Data Management and Migration

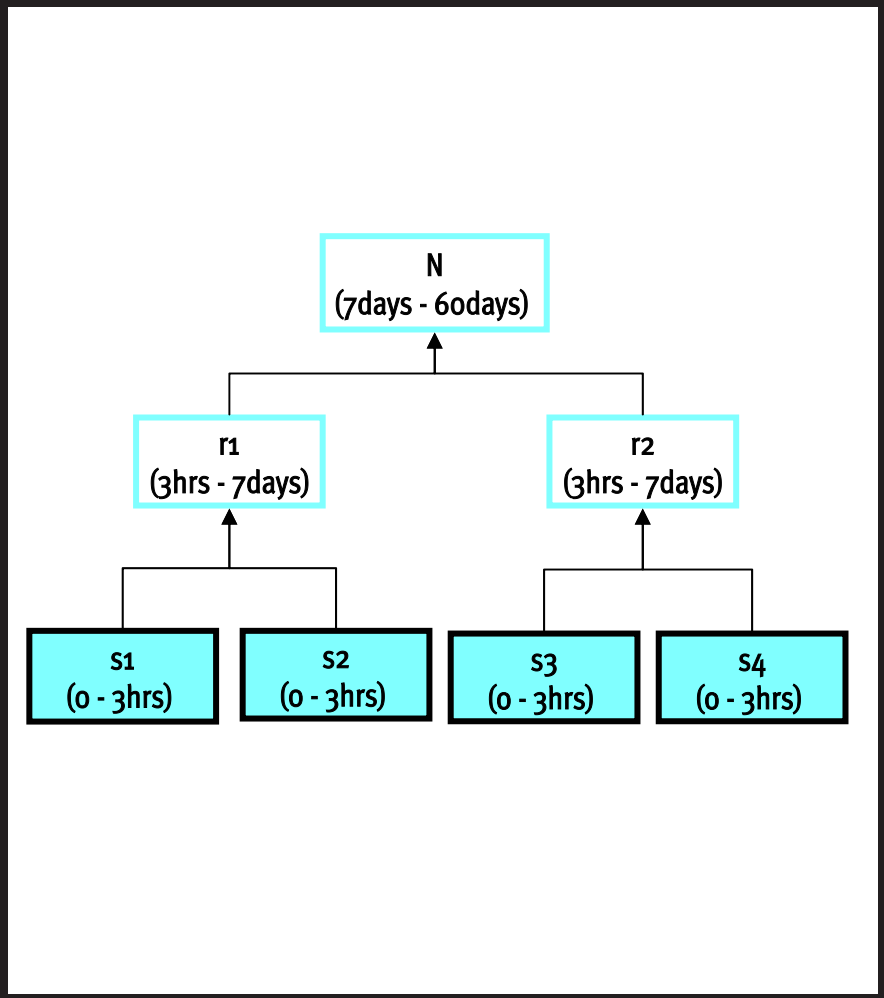


- EPC data is collected by the Edge Savants and is migrated to the Internal Savants.
- EPC data is distributed among various nodes of the savant network based on two attributes

  Time: As time goes by the data collected by a particular Savant can be found as we travel vertically in the network

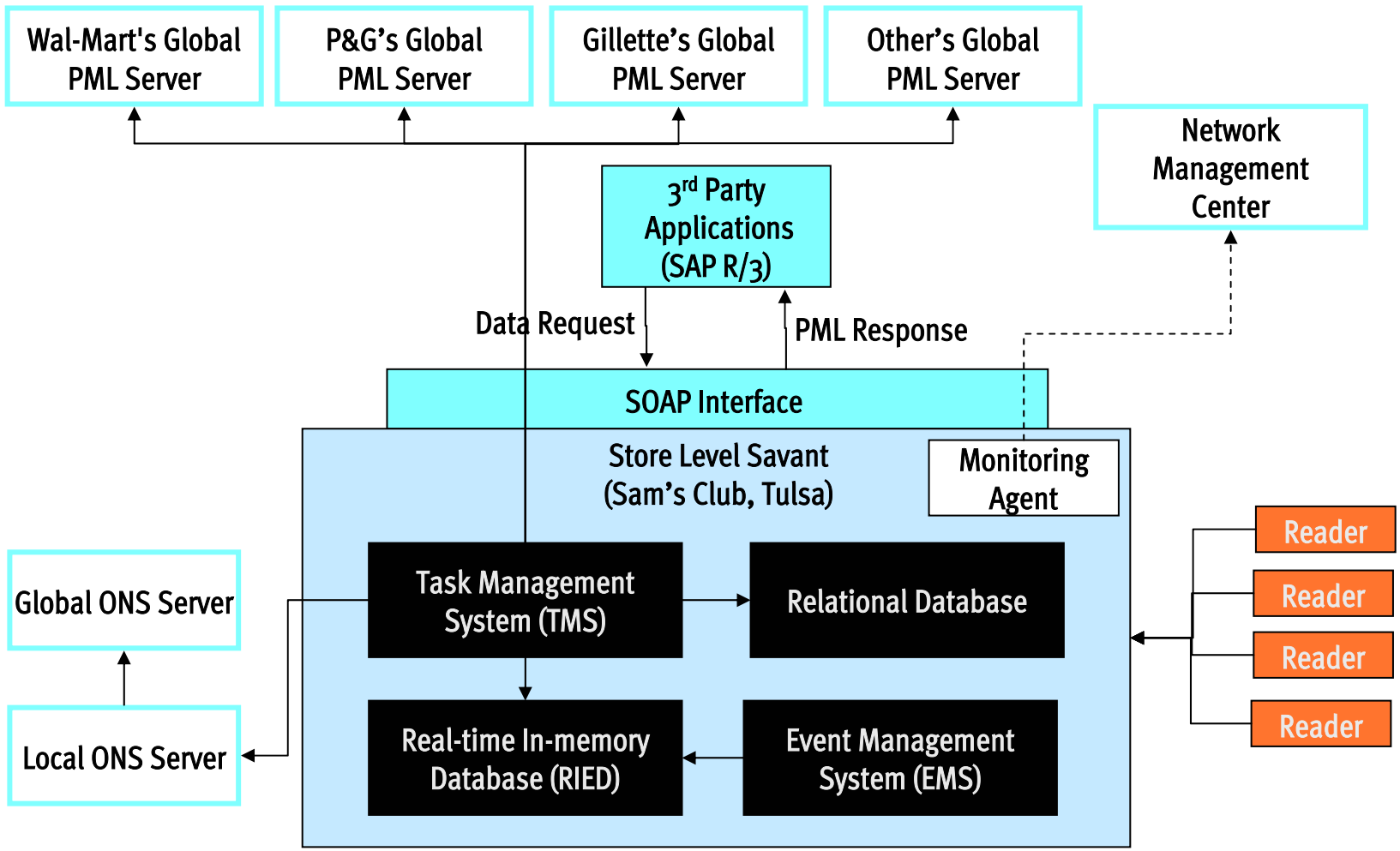
  Space: Data collected by a particular Savant can be found only on the Savants that are on the path from the root of the tree to it.

Localized Data Monitoring



- Savant is a self-replicating system
- Real-time EPC data can be monitored by the Edge Savants capturing that data
- Aggregated EPC data can be monitored by the Internal Savants

Components of the Savant



Components of the Savant

- Event Management System (EMS)

  Capture, filter, broadcast & log EPC data in real-time

- Real-time In-memory Event Database (RIED)

  Maintains the latest tags read by various readers which can be queried through a native/JDBC interface

- Task Management System (TMS)

  Schedules and executes tasks. These tasks can be used for data monitoring and migration

Event Management System (EMS)

- Event Management System captures, filters, broadcasts & logs EPC data
- Event Management System is implemented on Edge Savants (ES) since the EPC data enters the Savant network only through the Edge Nodes

Requirements of EMS

- EMS should be a high-performance system
- Support multiple EPC readers that communicate using different Auto-ID center protocols
- Support event filters with one input stream and multiple output streams for:

  Smoothing, Coordination, Forwarding
- Support for multiple loggers

  Database, Memory model, remote server (HTTP, SOAP, JMS)
- Ability to handle spikes in the event volume

Components of EMS

- Reader Adapter

  Communicates with the readers to capture EPC events

- Event Queues

  Asynchronous queuing system that captures EPC data from various reader adapters and broadcasts the data to multiple event loggers

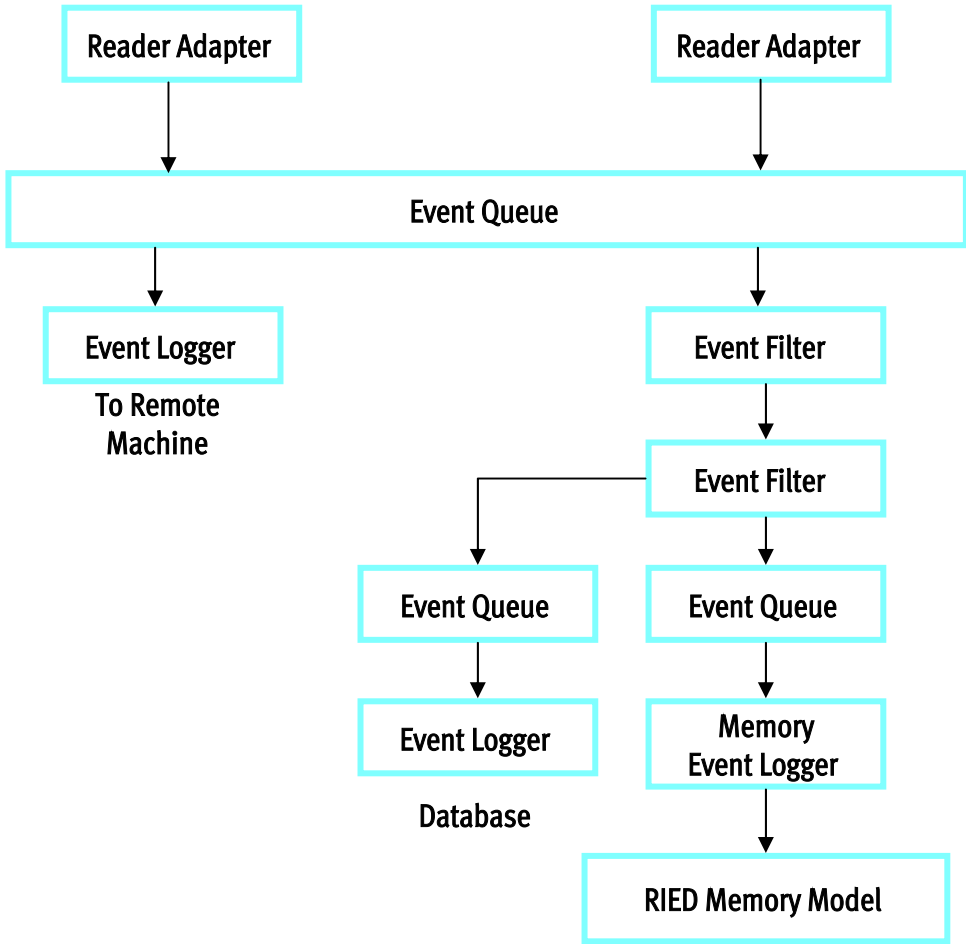
- Event Filters

  Gets data from one or more input event streams and posts the data to multiple output streams, after filtering the data.

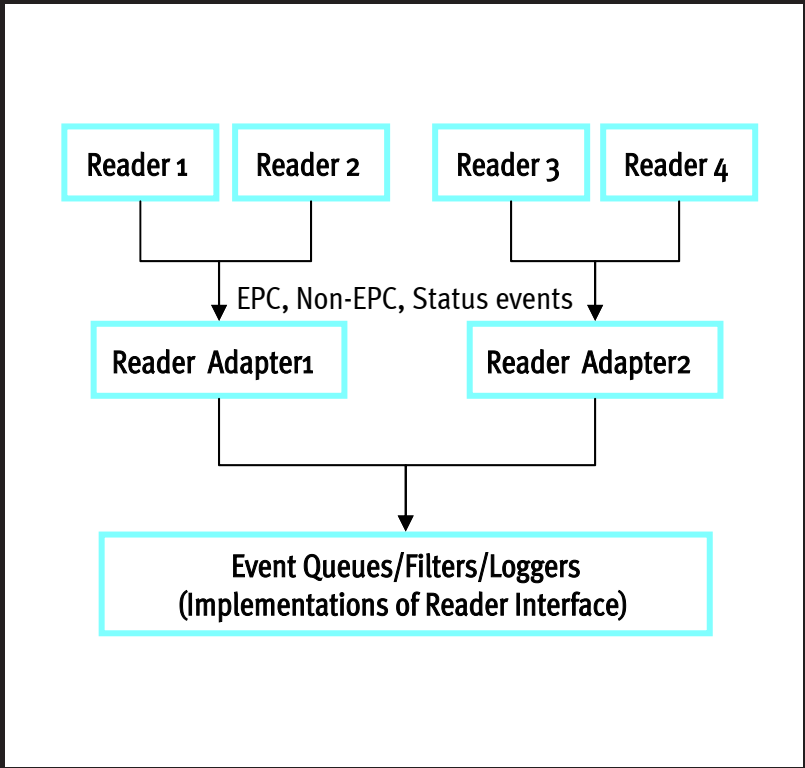
- Event Loggers

  Logs data to a database, memory model or a remote server (HTTP, SOAP, JMS)

Sample EMS

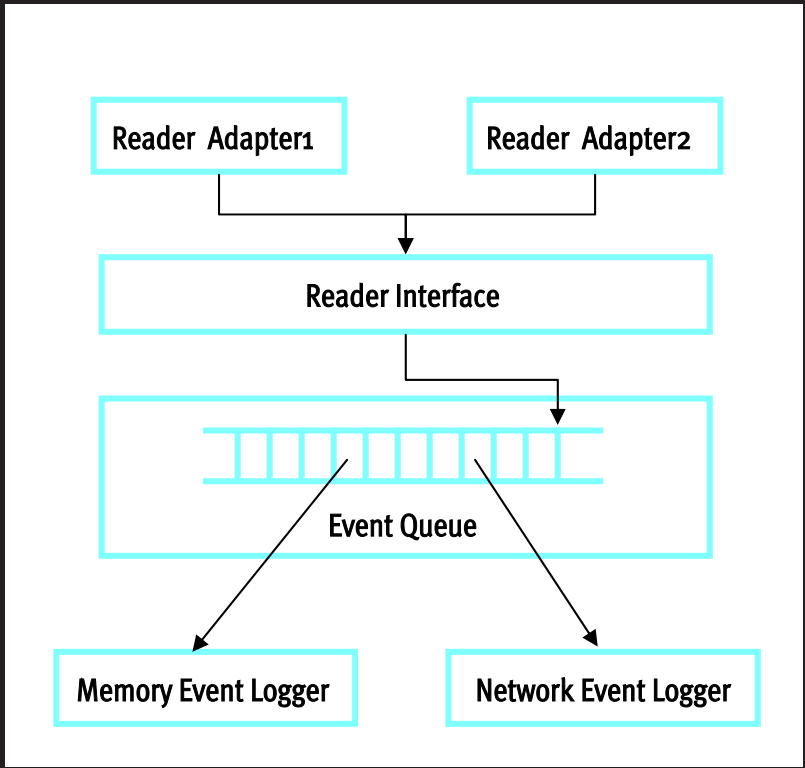

Reader Adapter
Reader Adapter
Event Queue
Event Logger
To Remote Machine
Event Filter
Event Filter
Event Queue
Event Queue
Event Logger
Database
Memory Event Logger
RIED Memory Model

Reader Adapter



- Reader Adapter communicates with the readers to capture EPC data
- Reader Adapter is an 'event producer' that posts event data to any 'event consumers' that implement reader Interface
- All modules which capture data from the readers have to implement **`ReaderAdapterInterface`** to be part of the EMS framework

Event Queue



- Buffers data to handle spikes in event volumes
- Asynchronous queuing system that captures EPC data from various reader adapters and broadcasts the data to multiple event loggers
- All event queues should implement **`Reader Interface`** to be part of the EMS framework

Event Filter

- Event Filters are added between event producers and consumers to Smooth, Coordinate and Forward event data
- Event Filters get data from one or more input streams and forwards data to multiple output streams, after filtering
- Event Filters can filter data based on

  EPC, EPC Prefix, Reader EPC, Reader Location, Time
- Event Filters should implement **`EventFilterInterface`** to be part of the EMS framework

Event Logger

- Logs or Broadcasts events
- Type of loggers

  Database logger: Logs events into a database

  RIED Logger: Logs events to real-time in-memory event database

  Network loggers: Logs events to remote servers. Can log these events through SOAP,HTTP, TCP/IP
- Event Loggers should implement `ReaderInterface` to be part of the EMS framework

3rd Party Applications Subscribing for EPC Data

- 3rd party applications can subscribe to the public event queues to get real-time EPC data
- 3rd party applications can register themselves through a SOAP interface
- The subscriber can choose an event filter that the data has to pass through before the queue broadcasts the data

- SOAP Services Installed

EMS SOAP Interface

- **public static String startup**()
- **public static void shutdown**()
- **public static Vector listPublicListeners**()
- **public static String addPublicListener**
  (String listenerName, String queueName,
  String loggerClass, String loggerArgs)
- **public static String addPublicListener**
  (String listenerName, String queueName,
  String loggerClass, String loggerArgs,
  String filterClass, String filterArgs)
- **public static void removePublicListener**
  (String listenerName)

EPCs in use for the Demo

00.0000001.000001.000000001 (Gillette – Mach3)

00.0000001.000001.000000002 (Gillette – Mach3)

00.0000001.000001.000000003 (Gillette – Mach3)

00.0000002.000002.000000001 (P&G – Covergirl)

00.0000002.000002.000000002 (P&G - Covergirl)

00.0000002.000002.000000003 (P&G – Covergirl)

Demo

- 3rd Party application subscribing to a public event queue
- Get PML Readings over SOAP using Remote Event Dispatcher

  All reads

  Only Gillette reads
- Get EPC event broadcasted in real-time

  All reads, file

  Only Gillette reads, file

EMS Implementation – Technology

- EMS is a pure-Java implementation (JRE 1.2+)
- ANTLR (ANother Tool for Language Recognition) parser generator for the EMS configuration parser
- Apache Tomcat SOAP server for the external EMS interface to add, monitor and remove remote listeners

EMS Implementation – Reader Adapters

- Implement the interface "`org.autoidcenter.epms.reader.comm.ReaderAdapterInterface`" This interface defines a shutdown method with no arguments or return value.
- Implement a constructor taking a `String`, and a `ReaderInterface` argument. EMS passes the initialization `string` specified in the startup clause of the adapter command as the first argument. It passes the EMS unit specified in the for clause of the adapter command in the second argument.
- Configuration

```
adapter intermec_adapter_1 is
    org.autoidcenter.ems.adapter.ImecAdapter startup
    "port=10101" for main_queue;
```

EMS Implementation – Event Loggers

- Implement the interface "org.autoidcenter.epms.reader.comm.ReaderInterface"
- Implement a constructor taking a `String` argument. EMS passes the initialization `string` specified in the startup clause of the logger command to this argument.
- Configuration

```
logger db_logger is
org.autoidcenter.ems.logger.DBEventLogger startup
"name=db_logger
db_url=jdbc:postgresql://localhost/epc_data
db_user=postgres db_password=postgres";
```

EMS Implementation – Event Filters

- Implement the interface "org.autoidcenter.epms.reader.comm.EventFilterInterface". On startup, the EMS invokes the `setListeners` method to assign the output processing units for this filter.
- Implement a constructor taking a `String` argument. EMS passes the initialization `string` specified in the startup clause of the filter command as this argument
- Configuration

```
filter memdb_filter is
org.autoidcenter.ems.filter.MemoryLoggerFilter
startup "reference_epc=00000000000000000000000
max_read_interval=1000" output (add_queue
remove_queue);
```

Performance Statistics

- Hardware & Software Configuration:

  DELL PowerEdge Server 2500, with 1133MHz Intel Pentium III processor, 512KB cache, and 512MB RAM, running Blackdown release Java 1.3.1

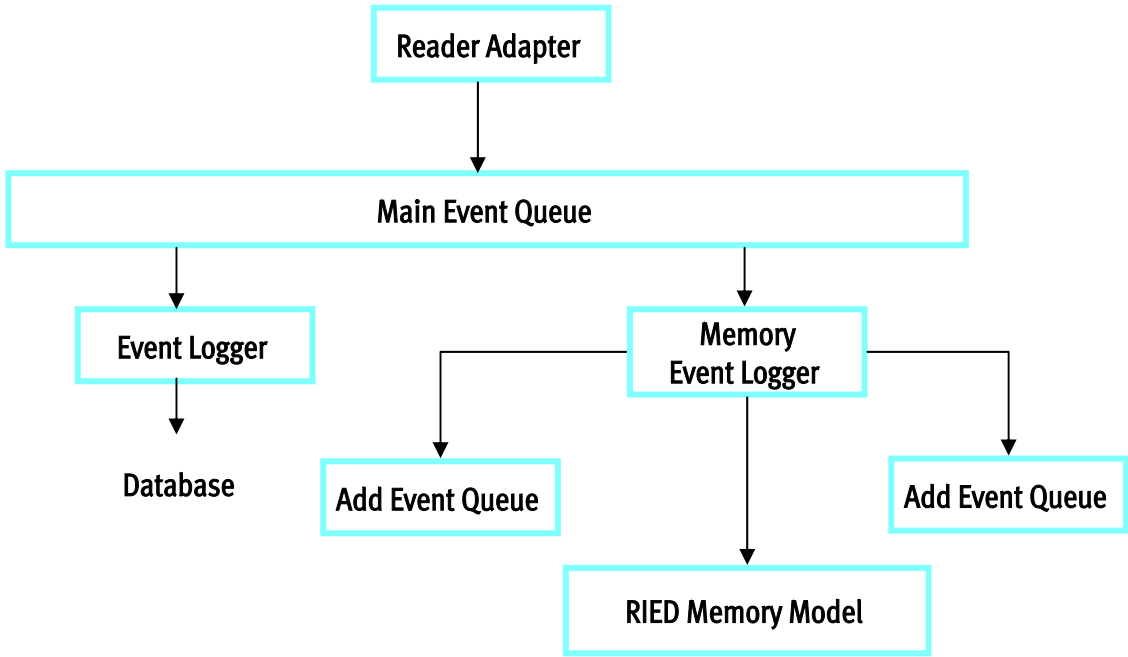
- Test Scenario

  The test involved sending 1 million events through an Event Queue of size 100K events. The Event Logger simply maintained the number events received

- Performance

  10.3 µs/EPC Event (~95,000 EPC Events/sec)

Demo Configuration



EMS Configuration file

EMS Configuration Grammar

Demo

- Attach the client to the Add Queue, file
- Attach the client to the Remove Queue, file

Real-time In-memory Event Database (RIED)

- Real-Time In-Memory Event Database is used to maintains latest event information by the Edge Savants

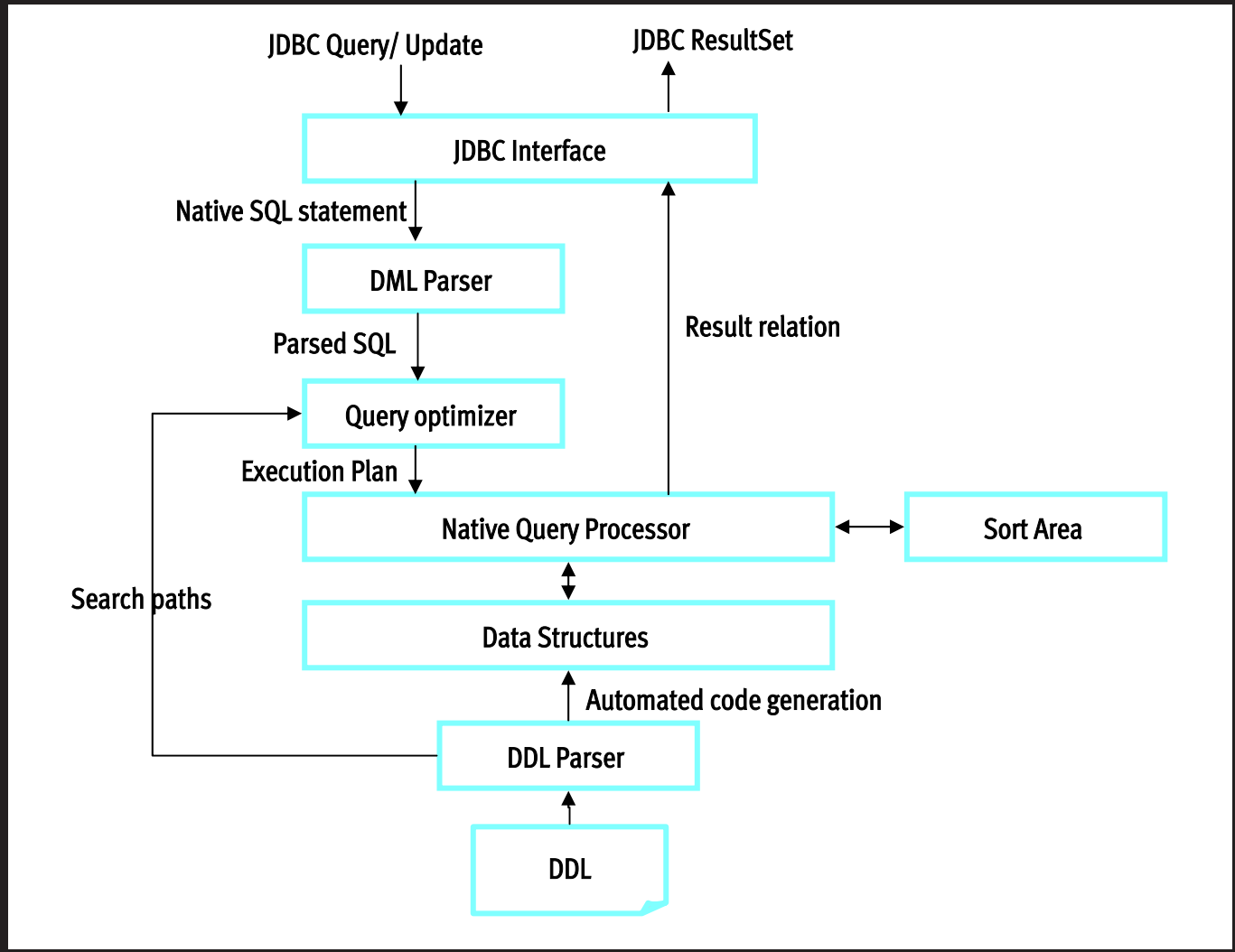
RIED - Requirements

- High-performance in-memory database
- Database should be able to maintain multiple snapshots
- Provide a standard API (JDBC) to access and manipulate data

RIED – Design Choices

- Reduced Data Manipulation Language (DML) complexity
- Reduced Data Definition Language (DDL) complexity
- Simple query optimization
- No constraint maintenance and triggers

RIED – Architecture



RIED Implementation - Technology

- RIED is a pure-Java implementation (JRE 1.2+)
- ANTLR (ANother Tool for Language Recognition) parser generator for the DDL and DML parsers.

RIED Implementation – Data Definition Language (DDL)

- In-Memory database schema is defined in DDL
- DDL parser loads the table definitions and creates appropriate data structures
- Table are defined using `create table` command

```
CREATE TABLE <table name> (
  <column name 1> <datatype 1> [PRIMARY KEY | INDEX]
  <column name 2> <datatype 2> [PRIMARY KEY | INDEX]
  ...
);
```

- Data types are supported by RIED:
  - VARCHAR(n): A string datatype with maximum length n.
  - NUMERIC(m, [n]): A numeric datatype with scale m, and precision n.
  - BIGINT: A Java Long datatype
  - INTEGER: A Java Integer datatype
  - DOUBLE: A Java Double datatype
  - FLOAT: A Java Float datatype
- DDL Grammar

RIED Implementation – Data Manipulation Language (DML)

- RIED DML supports a subset of SQL92
- DML statements can be executed on the RIED system over a native Java interface, or using a JDBC driver implemented using Java RMI (Remote Method Invocation).
- SQL functions supported by RIED

  MIN, MAX, COUNT, SUM, ABS, LENGTH, TRUNC, ROUND, MOD, STRPOS, LOWER, UPPER
- New functions can be added to the RIED DML language easily by extending the `FunctionManager` class in the package `org.autoidcenter.memdb`

RIED Implementation – Data Manipulation Language (DML) – SQL Functionality

- SELECT

```
SELECT [DISTINCT | ALL] expr1, expr2, ...
FROM table1 [AS alias1], table2 [AS alias2], ...
 [WHERE select_expr1 ]
 [GROUP BY group_expr1, group_expr2, ... ]
 [HAVING having_expr ]
[[UNION | EXCEPT | INTERSECT ] [ALL]
  another select query] ...
```

- INSERT

```
INSERT INTO table [(col1, col2, ...)] subquery
```

- UPDATE

```
UPDATE table
  SET col1 = expr1, col2 = expr2, ...
  [WHERE select_expr]
```

- DELETE

```
DELETE FROM table [WHERE select_expr]
```

RIED Implementation – Data Manipulation Language (DML) – SQL Functionality

- Specify table prefix to reference a column

  "`SELECT foo.bar FROM foo`" instead of "`SELECT bar FROM foo`"
- Use `COUNT` instead of `UNIQUE & EXISTS` keywords
- Complete DML grammar

RIED Implementation – Query Optimization Algorithm

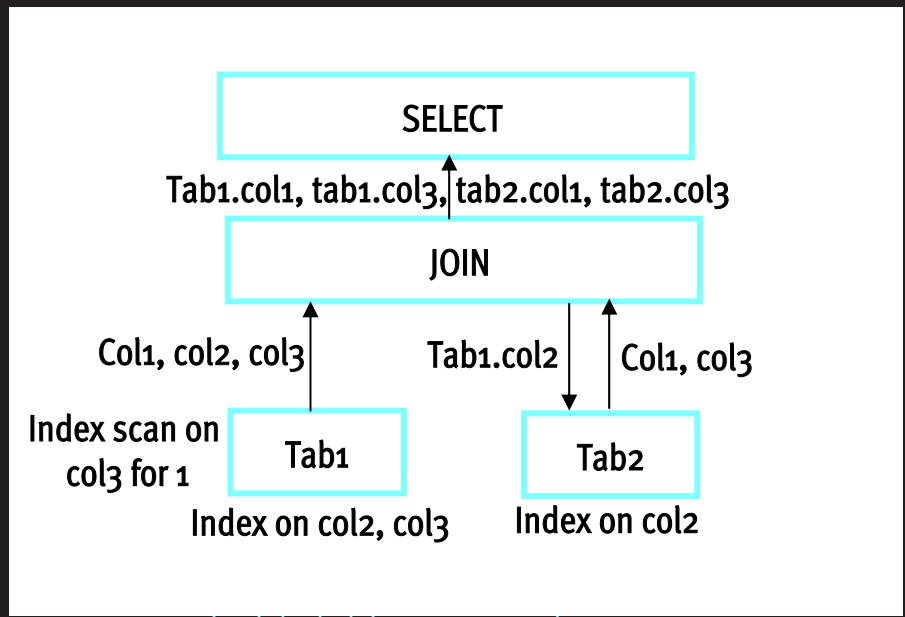
- RIED has a built-in query optimization algorithm

  For Example:

  SELECT tab1.col1, tab2.col1,

  FROM tab1, tab2

  WHERE tab1.col2=tab2.col2 AND tab1.col3=tab2.col3 AND tab1.col3='1'



AUTO-ID CENTER

RIED Implementation – Version Maintenance

- RIED uses Sequence Manager for version maintenance
- RIED supports COMMIT and ROLLBACK functionality
- Currently RIED supports one transaction at a time on the latest version, this is mainly because of the high-cost of lock and unlock operations on records
- RIED supports multiple parallel connections on previous snapshots

HOW TO ACCESS RIED

- RIED can be accessed through

  Native Interface

  JDBC Interface

Demo

- Attach the client to the Add Queue, file
- Attach the client to the Remove Queue, file

Performance Statistics – Persistent Database

- Hardware & Software Configuration:

  DELL PowerEdge Server 2500, with 1133MHz Intel Pentium III processor, 512KB cache, and 512MB RAM, running Blackdown release Java 1.3.1, PostgreSQL (7.0.2)

- Test Scenario

  Test involved sending 100K events to a database logger. The database already contained 200K events when the test started. Every event was logged in the observation table. The latest observation for each EPC is maintained in a parent table called object. (Schema)

- Performance:

  10 ms/event logged (100 events/second)

Performance Statistics – RIED

- Hardware & Software Configuration:

  DELL PowerEdge Server 2500, with 1133MHz Intel Pentium III processor, 512KB cache, and 512MB RAM, running Blackdown release Java 1.3.1, PostgreSQL (7.0.2)

- Test Scenario

  Every event was logged in the latest_epc_observation table. This logger performs "smoothing" by associating each object EPC to exactly one reader EPC at any time. Any read from a different reader is logged only if the latest timestamp entry for that EPC is older than 2 seconds. (Schema)

- Performance:

  66.5 µs/event logged (15,000 events/second)

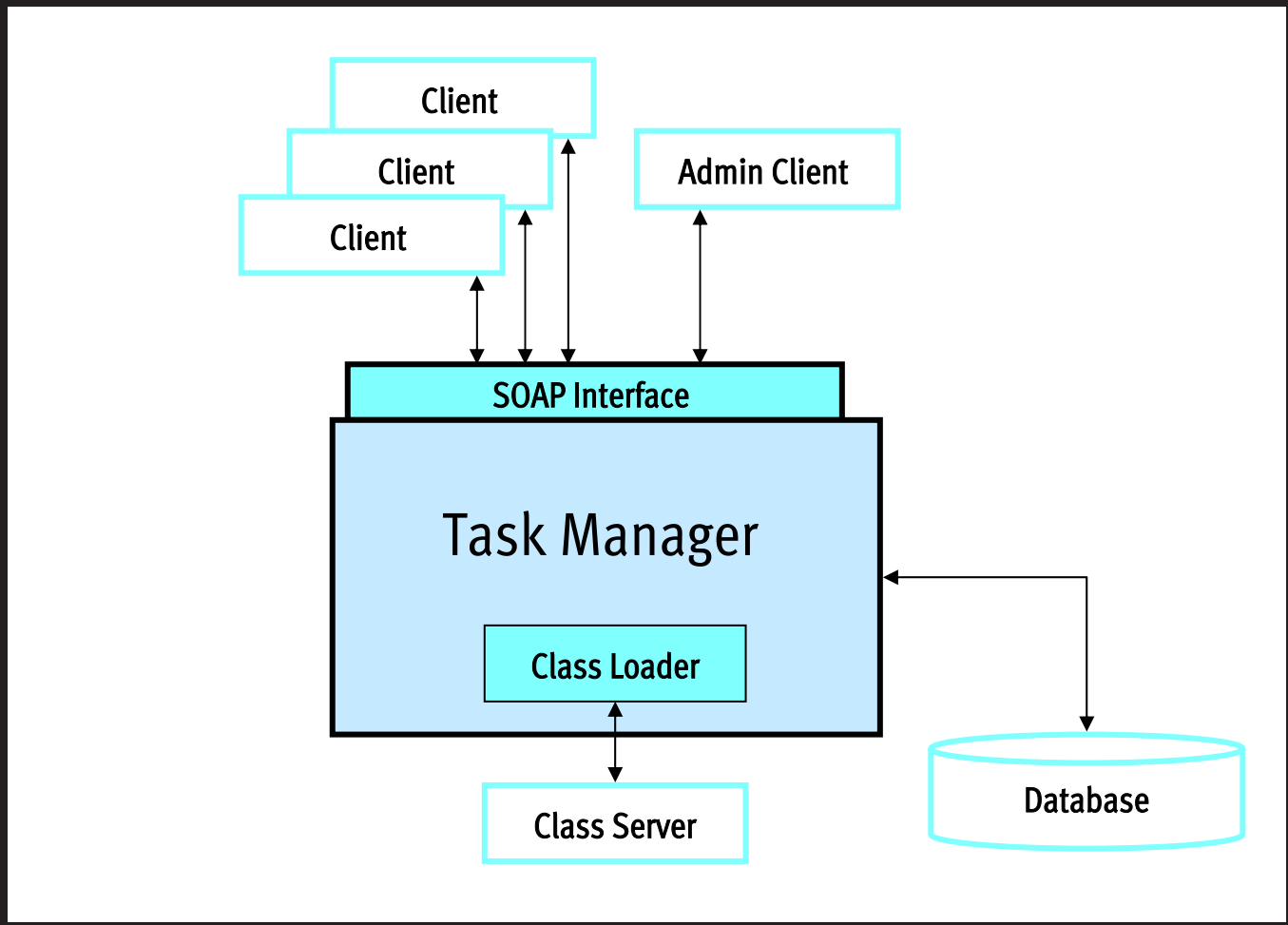
Task Management System (TMS)

- The Savant performs data management, and data monitoring using customizable *tasks*
- Task Management System (TMS) manages tasks as operating system manages processes

TMS – Requirements

- The Savant TMS should be a platform-independent system requiring little memory processing power
- The Savant TMS should automatically upgrade the tasks it executes
- The Savant TMS should present a well-defined, interoperable external interface to schedule, monitor, and remove tasks
- Tasks should be written in a platform-independent language using a simple, well-defined SDK

TMS - Design


Client
Client
Client
Admin Client
SOAP Interface
Task Manager
Class Loader
Class Server
Database

AUTO-ID CENTER

Task Manager

- Task Manager is responsible for running and maintaining tasks running on a savant on behalf of user
- Every task submitted to the system consists of a schedule, which determines how often the task must run, whether it should be continuously running, etc.
- Depending on the schedule, the Task Manager attempts to resolve which task to run at each time interval

AUTO-ID CENTER

Types of Tasks

- One-time task

  The Task Manager spawns the query task and returns the result.

- Recurring task

  The Task Manager maintains the recurring schedule in a "persistent store" and then executes the task given the schedule.

- Permanent task

  This task is executed continuously by the Task Manager. The Task Manager periodically monitors the task and in case of any failure the Task Manager re-spawns the task.

SOAP Interface

- startup: Startup the TMSTaskServer.
- shutdown: Shutdown the TMSTaskServer.
- getPermanentTask: Retrieve a permanent task.
- getRecurringTask: Retrieve a recurring task.
- getAllPermanentTasks: Retrieve all permanent tasks.
- getAllRecurringTasks: Retrieve all recurring tasks.
- addRecurringTask: Add a recurring task.
- addPermanentTask: Add a permanent task.
- addOneTimeTask: Add a one-time task to the system.
- removePermanentTask: Remove a permanent task.
- removeRecurringTask: Remove a recurring task.

Class Server

- Class Server maintains the latest version of the tasks that are used by the Task Manager
- Task Manager maintains a reference to the Class Server and loads the new classes on demand when they become available without any restarts

TMS Implementation – Technology

- TMS is a pure java implementation (JRE 1.2+)
- Apache Tomcat SOAP server for the external TMS interface to view, schedule and remove tasks.
- PostgreSQL Database for the TMS persistent store.
- Apache Tomcat Webserver for the administration utilities.
- Apache HTTP server for the class server implementation.

TMS Implementation – Writing Task

- Implement the interface `org.autoidcenter.tms.TaskInterface` for recurring or permanent tasks
- Implement the interface `org.autoidcenter.tms.OneTimeTaskInterface` for one-time tasks
- Implement a constructor taking one arguments, the String input data for the thread
- Place the class in the class servers accessed by the Savant TMS or in the CLASSPATH
- Pass the appropriate `String` input data when executing the task using the TMS SOAP interface.

AUTO-ID CENTER

Task Manager Algorithm

- The Task Manager behaves exactly as the cron scheduler when handling recurring tasks..
- As for permanent tasks, the Task Manager checks if all permanent tasks are running in the system every minute. If a task is down, it is automatically re-spawned.
- The Task Manager ensures that at most one instance of every task is running in the system at any given time. That is, it does not re-spawn a task until the previously scheduled instance is done executing.

Simple Tasks

- ONS/PML Lookup Task

  This recurring task uses the ONS and PML servers to locate information about objects encountered by the Savant. The product information is then cached in the database. Task

- Shelf Stock out Task

  This task continuously monitors the shelf inventory of a certain product and sends alerts when the inventory drops below certain threshold. Task

- Data Migration Tasks

  These recurring tasks export EPC data up the logical Savant tree.

  Leaf node, Parent node

- SQL Query Task

  This is a one-time task that performs an SQL query and returns the results in PML

Future Work

- Remote monitoring & maintenance of the Savant and all its components
- Remote software upgrades of the Savant and key tools (JRE, Apache)
- Provide RIED snapshot management system
- Security infrastructure

Questions?

Contacts

- Prof. Sanjay Sarma (sesarma@mit.edu)
- Dr. Dan Engels (dragon@lcs.mit.edu)
- Prasad Putta (lprasad@alum.mit.edu,. 617-335-0693)