IBM – Haifa Research Lab | Global Business Services | IBM Israel

A Scalable Heterogeneous Solution for Massive Data Collection and Database Loading

Presented By **Uri Shani**

BIRTE'2006 – Seoul, Korea September 11, 2006

Authors

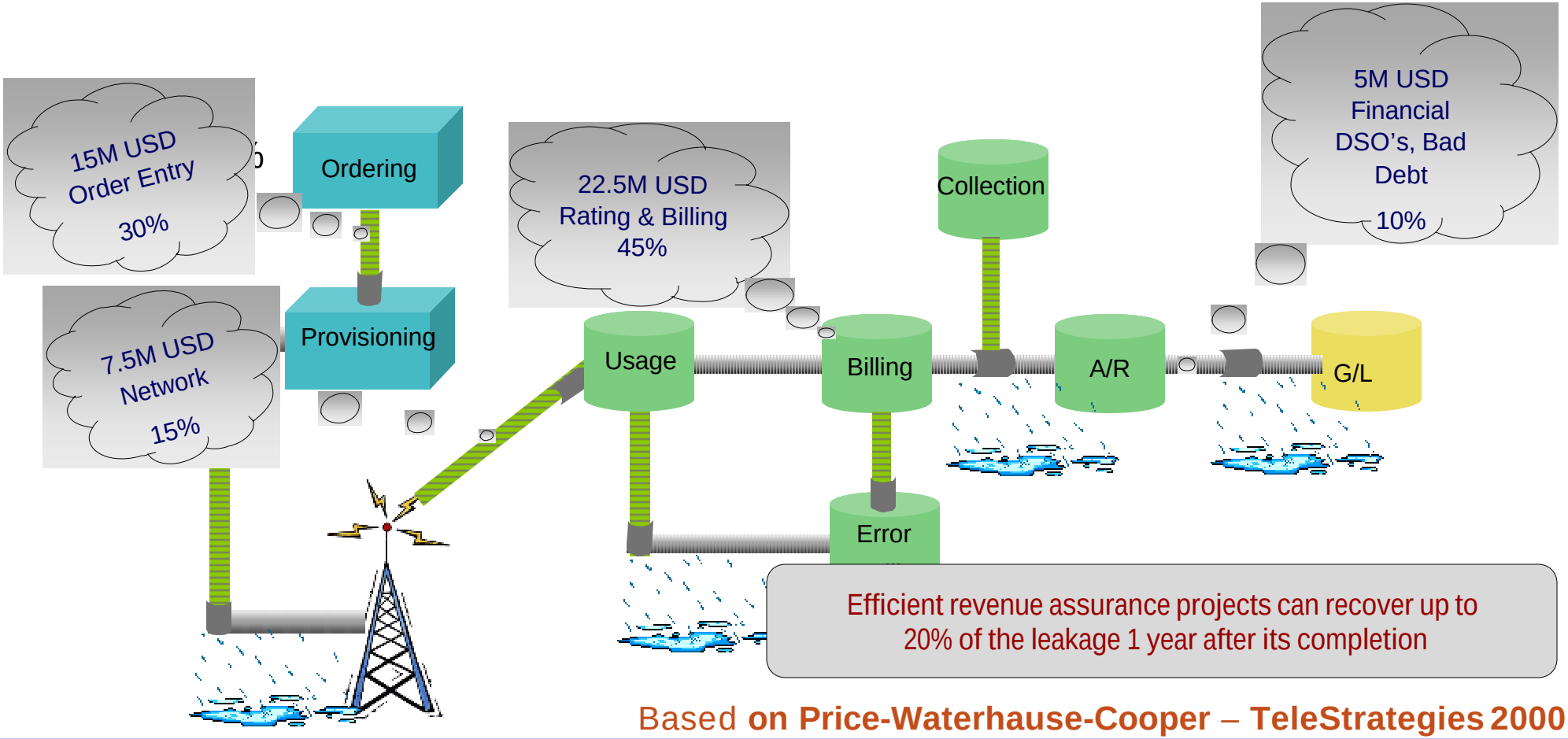
- IBM Haifa Research Lab (HRL)
  - Akilov A. Alex
  - Skarbovsky Inna
  - Sela Aviad
  - Shani Uri
- IBM Israel Global Business Services (GBS)
  - David H. Berk

- Acknowledgements
  - Yoel Arditi – GBS
  - Vortman Pnina – HRL
  - Dagan Gilat - HRL

Problem domain at large – Telco Revenue Assurance

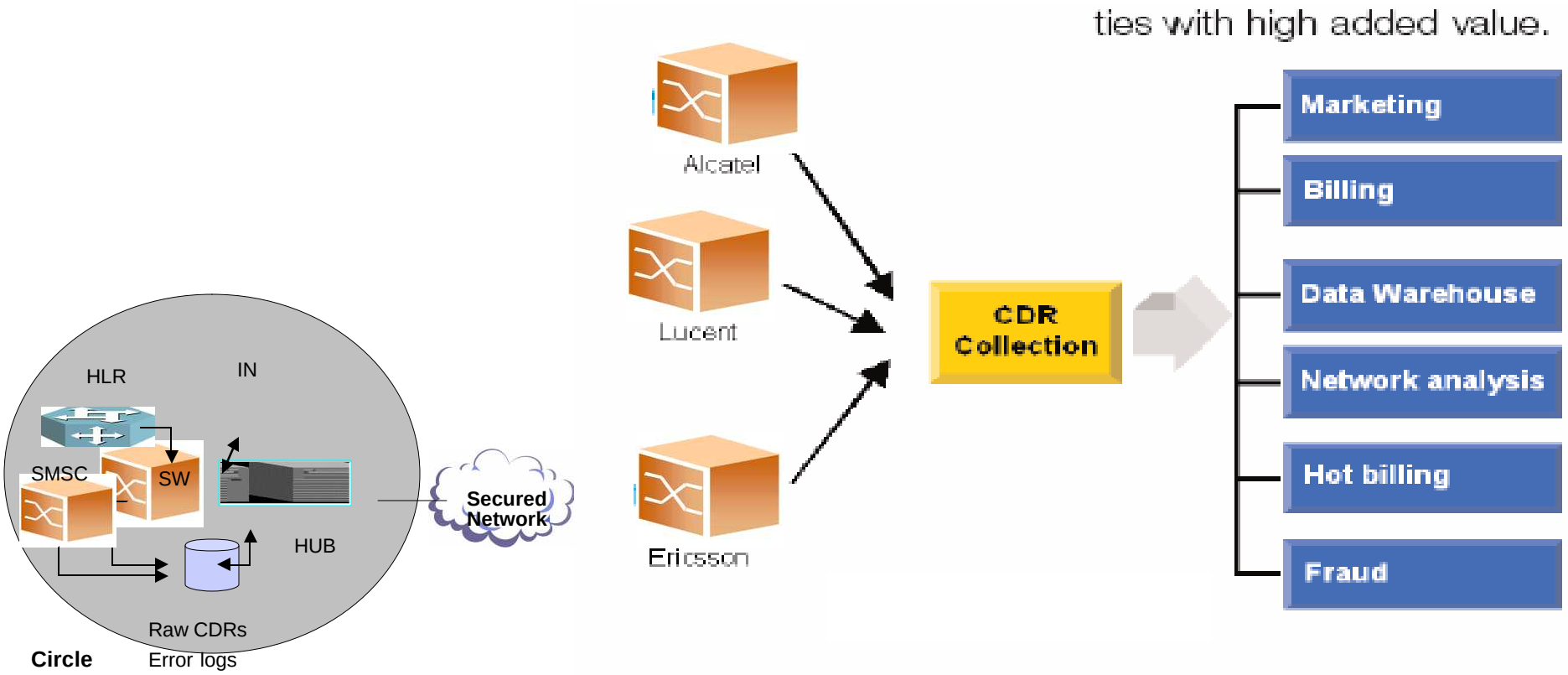
Illustrative example of potential RA Financial Benefits

Client Based Example: 50M USD out of 2.5B revenue (~2.5%)



Based **on Price-Waterhause-Cooper – TeleStrategies 2000**

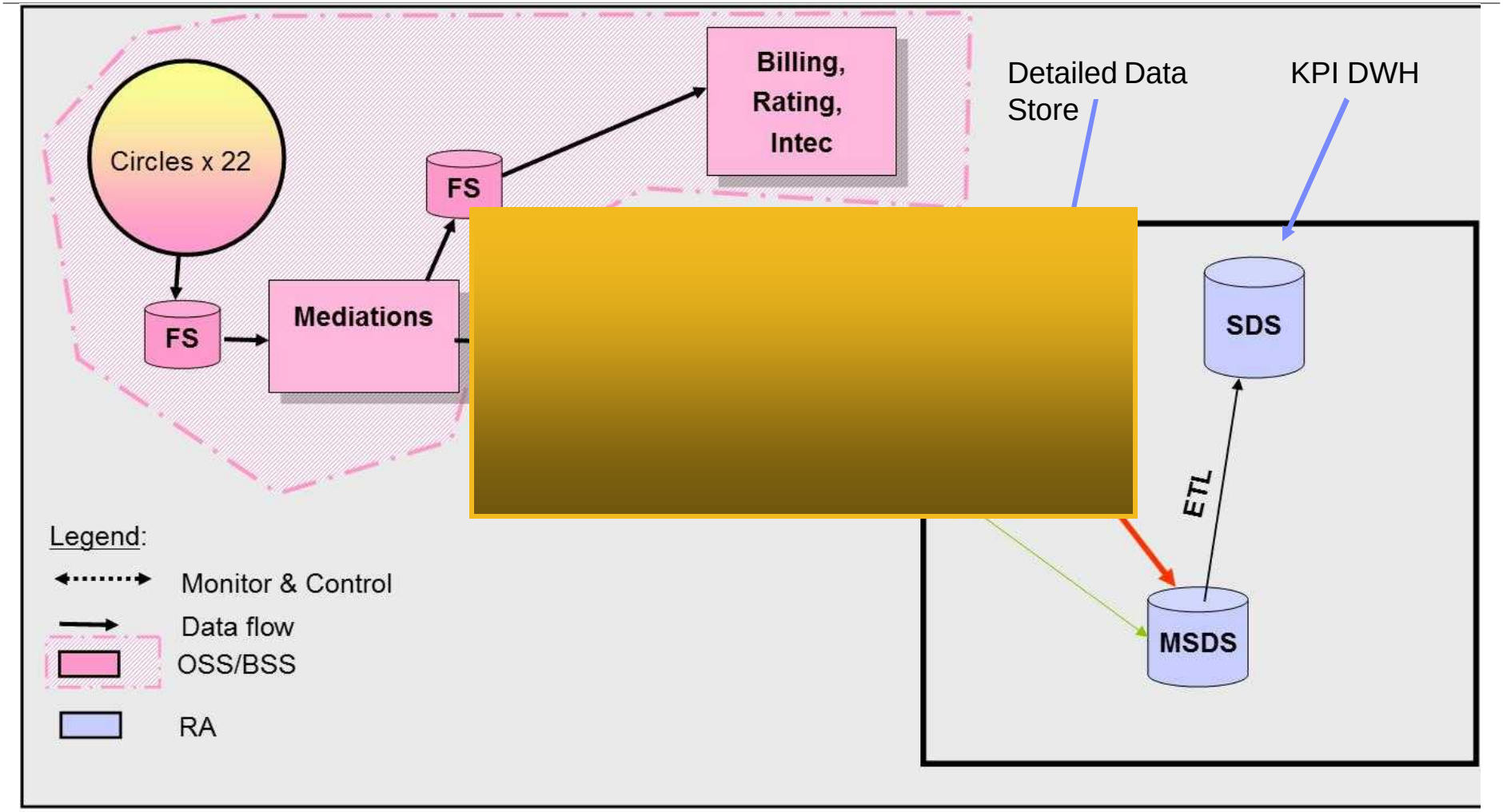
Telco Applications of CDRs



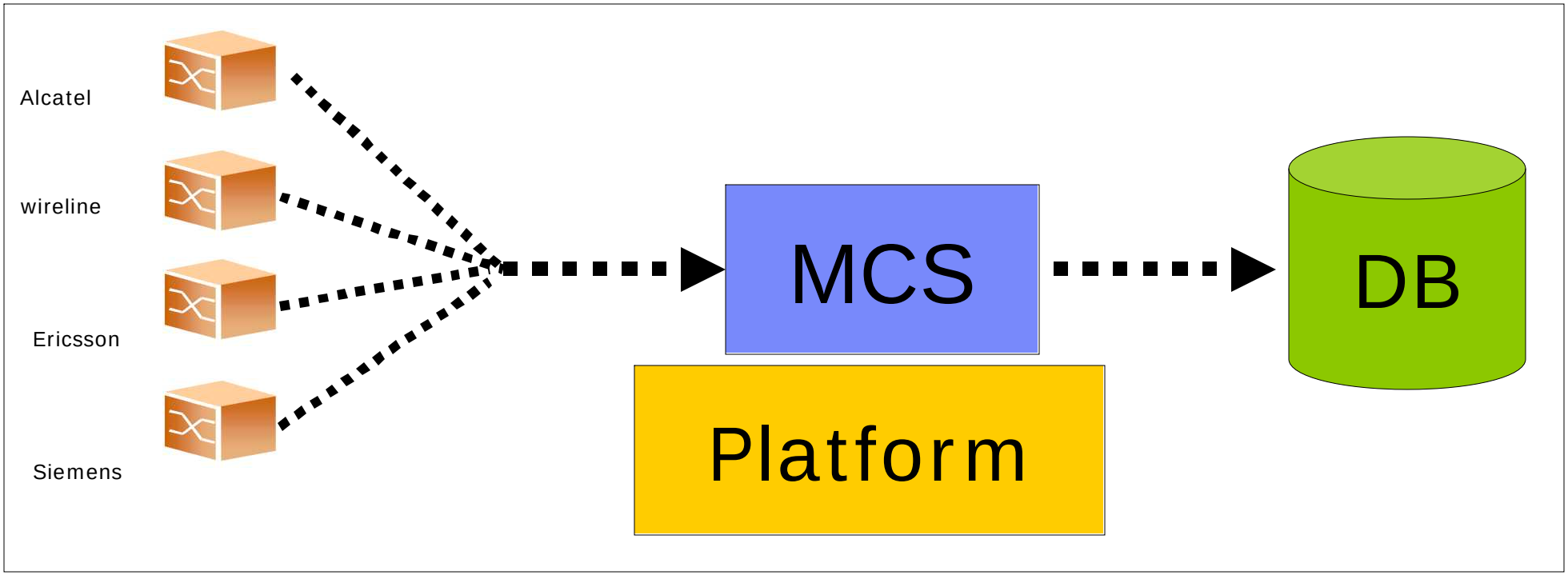
Problem Statement

- Problem at large
  - Revenue assurance
    - Business area: Telco
  - Two components:
    - Data warehouse of detailed CDRs – for drilldown
    - Data warehouse of key performance indicators (KPIs) – for discovery of discrepancies, using traditional OLAP methods, cube management and graphical dashboard by end-users.
  - Sources of data:
    - CDRs from Telco switches from geographically-spread circles
  - Tools:
    - ETL – (Extract Transform Load) tools
    - MCS (Massive Collection System) – subject of this project.
- Problem we deal with
  - Massive loading of CDRs to detailed CDR data-warehouse (to be used for drilldown)
    - While loaded, data is also screened for key performance indicators (KPI) for the KPI warehouse.
  - Input data is post mediated and unified using an ETL tool.

RA Components Architecture



MCS – Bird's eye view



MCS requirements - Functional

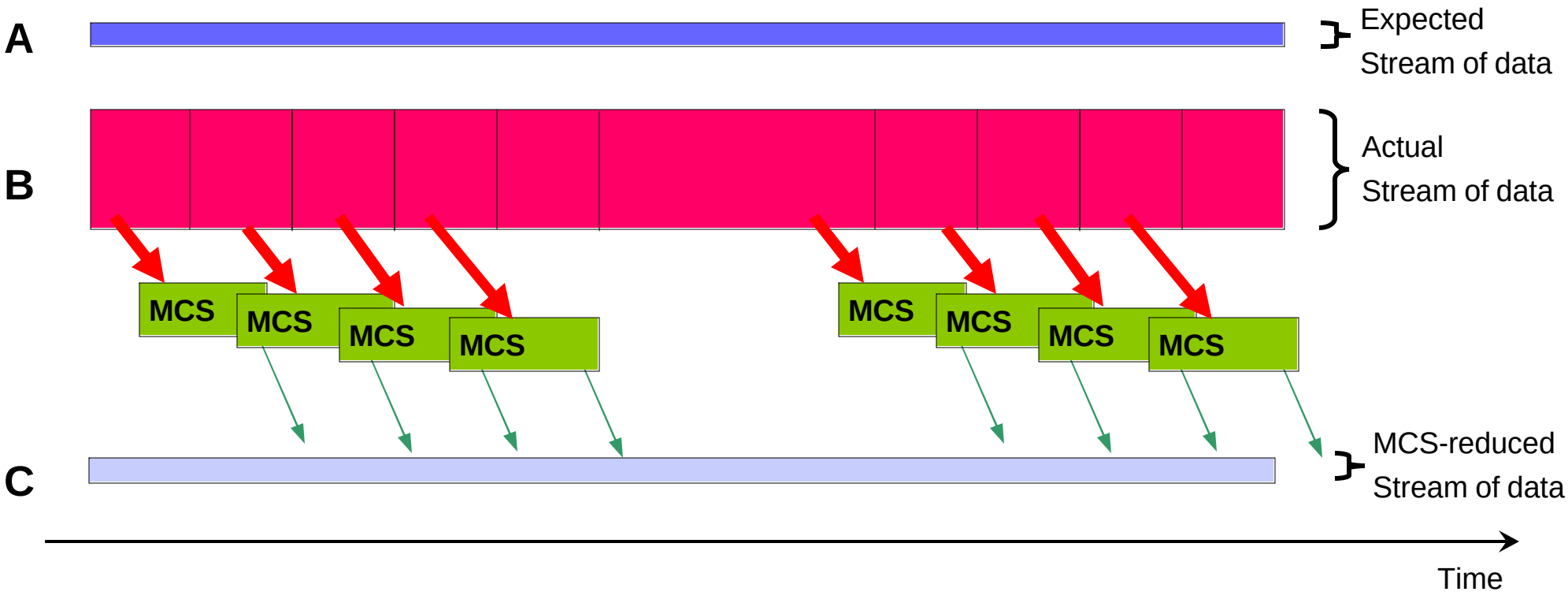
- Input
  - Stream of different kinds of CDRs, broken to files
  - Files have controllable number of records, ASCII format of fixed fields
  - Record size: 1-3KB, ~100 fields.
- Processing
  - Input is transformed according to a “Processing Plan”. This includes:
    - Format conversions
    - Dates calculations
    - Table lookups for conversions and membership tests
    - Gap analysis – find missing records
    - Unique key computation
      - Duplicate filtering
    - In general – prepare data for upload and KPI screening
- Output
  - Records ready for loading to DBMS
    - Records size: 1-3KB, ~100 fields
  - Loading these records to DDS tables and invoke KPI screening
  - → Management of the loading process
- Process management

MCS requirements - NonFunctional

- Input size
  - Initial estimate: 500 million CDRs/day
  - Expected growth: 50-60% annually
- Processing
  - Processing plan is easy to write by domain experts
  - Scalable
  - Transaction controlled
- Output
  - Scalable
  - Transaction controlled
- General
  - Scalable
  - Transaction controlled
  - Recoverable
  - High availability

MCS Operational principle

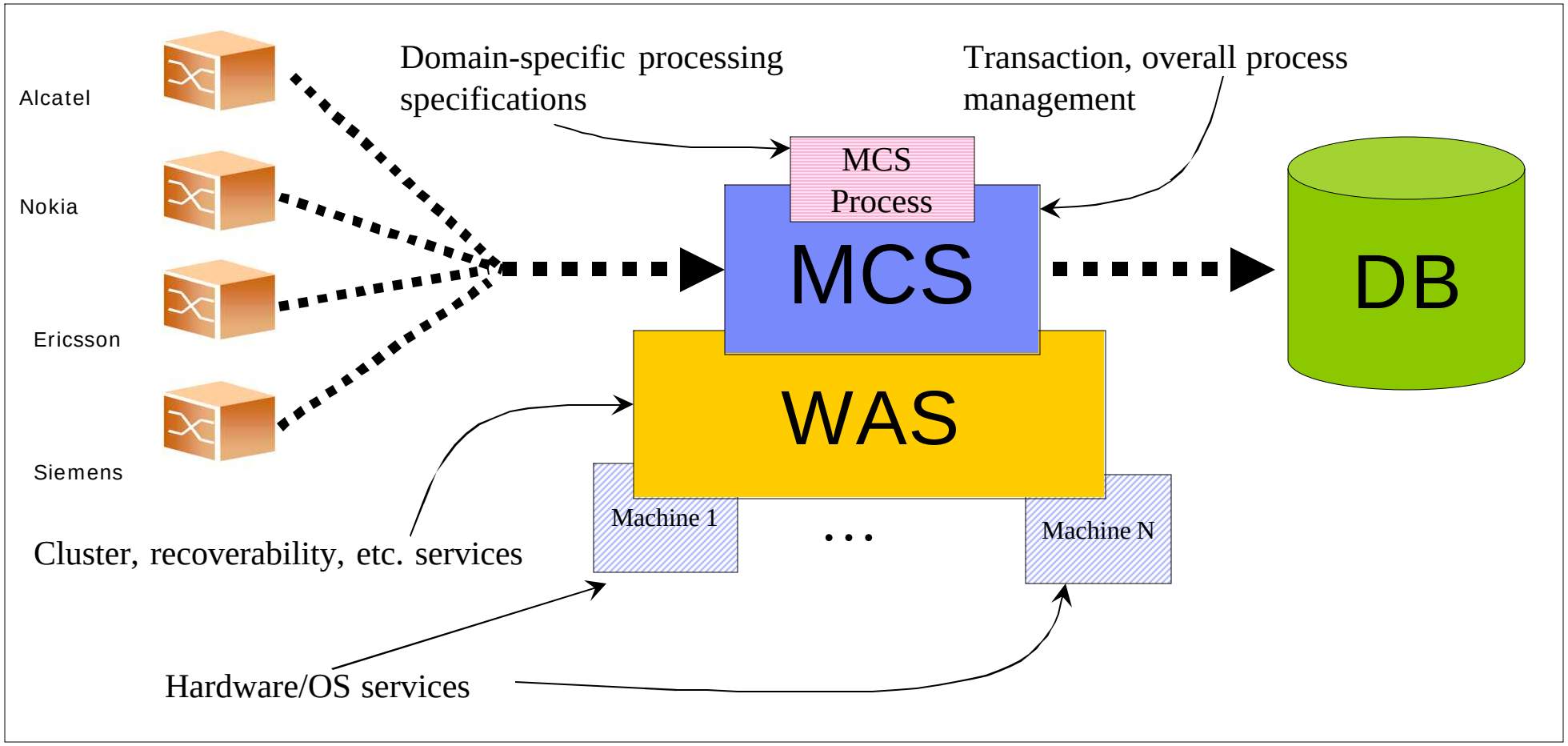
- Input stream broken to files with concurrent execution of processing plans



MCS Scalable Architecture – Infrastructure

- Using Websphere Application Server (WAS)
  - IBM's J2EE product
  - Using clusters of servers over multiplicity of machines (nodes)
    - →Scalability with load balancing
  - High availability with failover
  - Container-managed transaction monitoring, covering:
    - Database transactions
    - Overall process flow
      - EJBs invocation
      - Messaging
- Using DB2 DPF (Data Partitioning Facility)
  - Division of labor for parallel execution
  - Scalability
  - MDC (Multi-Dimensional Clustering)
    - Improves I/O and cache management for cube-oriented data
- Using GPFS file system (General Parallel File System)
  - Concurrent access to shared file system by all participating machines
    - Minimal overhead – practically negligible

MCS Scalable Architecture over Websphere Application Server



MCS Implementation Components

- High-level application execution framework on top of WAS J2EE architecture
  - Simplifies and practically hides most J2EE programming concerns
  - Preserves all WAS enterprise grade features
- Domain-specific language
  - Definition of the language
  - Eclipse-based development environment
  - Execution environment

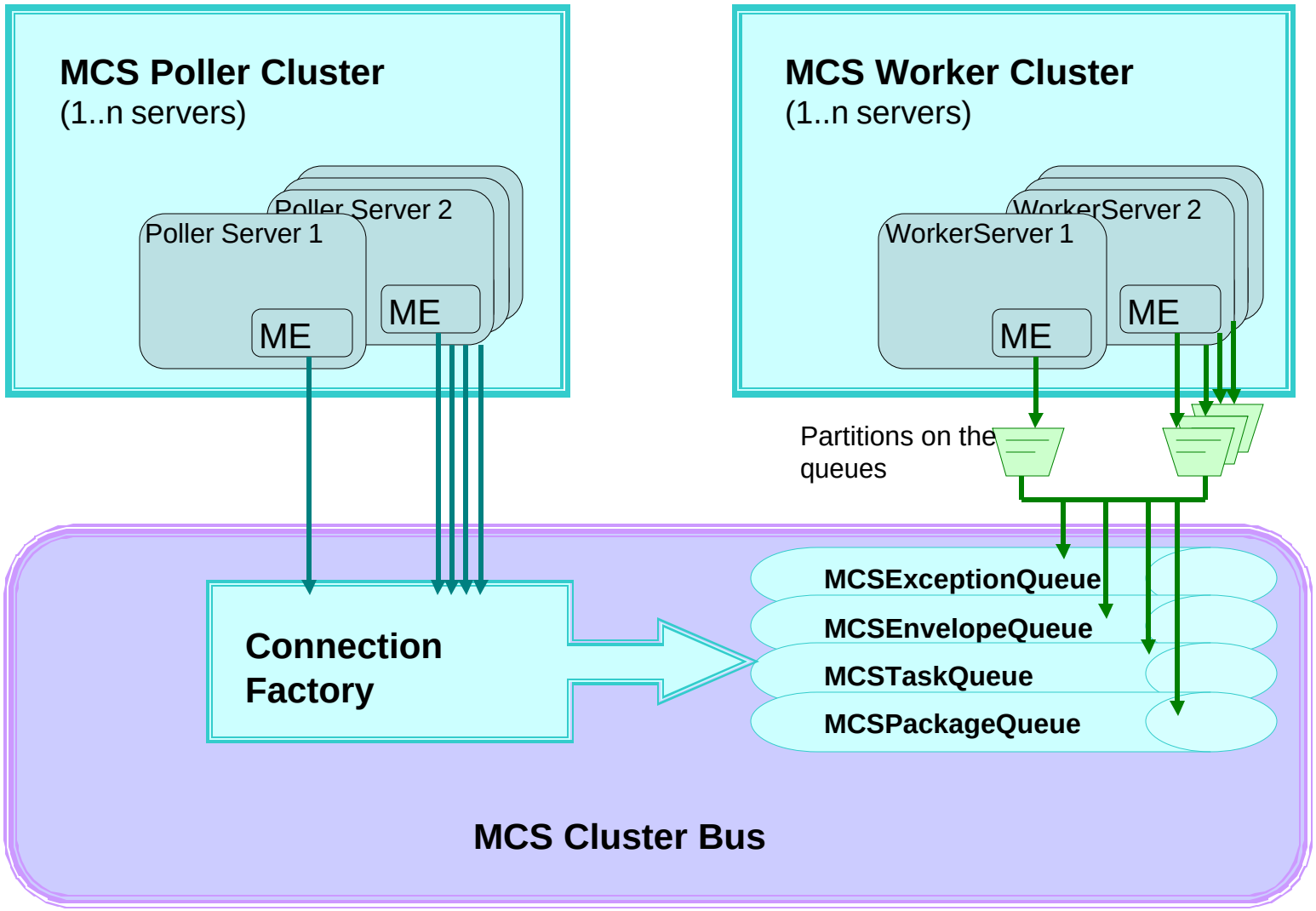
MCS Implementation Components – WAS side

- High-level application execution framework on top of WAS J2EE architecture
  - State machines to control long-term and short-term computational processes
    - Long term:
      - Poller – an activation component which polls for new files and activates processing
      - Package – a database loading agent which allocates ready output files and loads data effectively to the database using the DB2 Load utility
    - Short term dynamic:
      - Tasks – a processing activity which operates on individual input CDR files
      - Envelope – a management and processing activity which manages a group of CDR files and perform post loading database processing.
  - State machine execution engine – the MCS BPFC (Business Process Flow & Control)
    - Interprets a State diagram
    - Driven by triggers
      - Easy change from synchronized to a-synchronized execution
  - Triggers
      - Messages → asynchronous interaction
      - RMI or direct Java calls → synchronous interaction

Linking MCS entities with WAS facilities

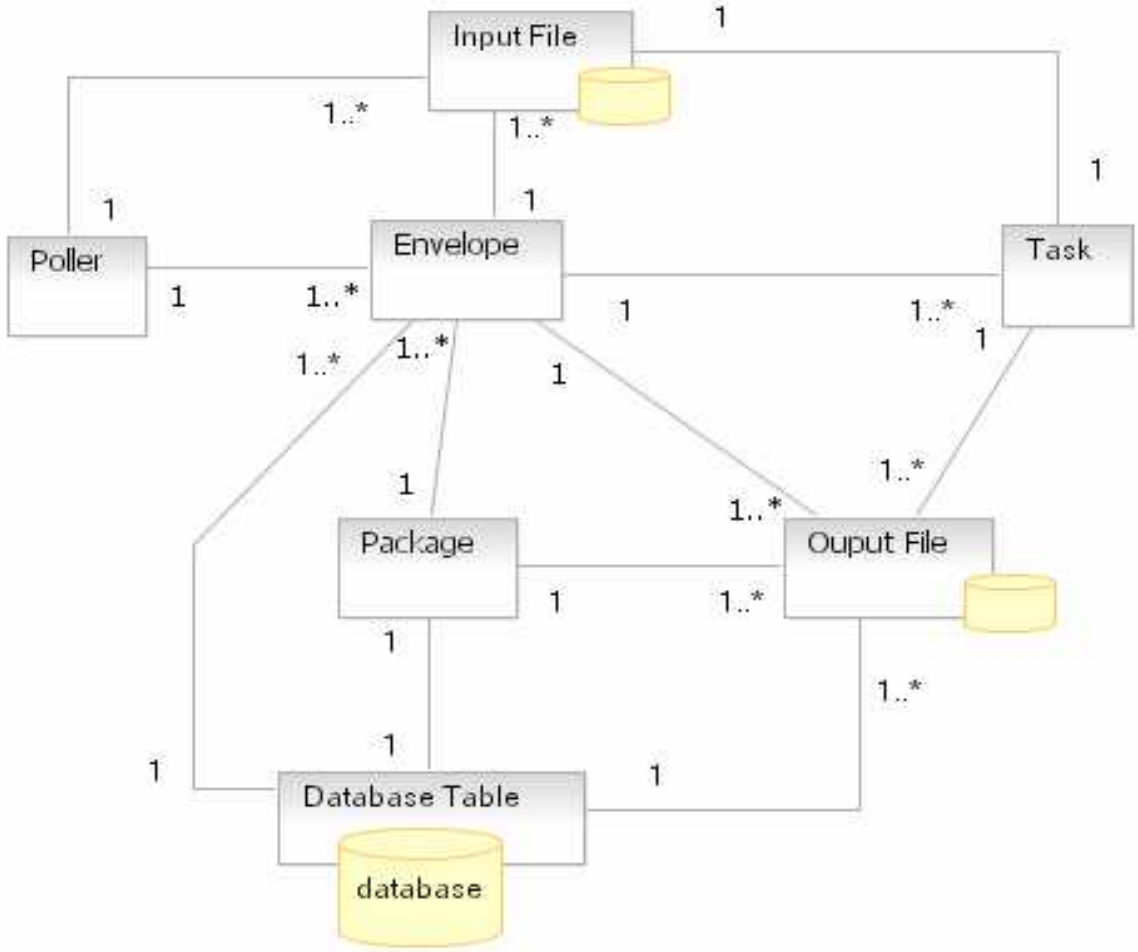
- State machine invocation via a message
  - Uses a J2EE MDB Message Driven Bean
  - Messages use internal messaging bus (ISB) of WAS
  - Allocates a thread from a thread pool
  - Concurrency is controlled by WAS deployment configuration
- State machine invocation synchronously
  - Uses simple EJB
  - Invoked via RMI over IIOP
  - Does not allocate new thread
  - Can be subject to container transaction control
- Database access via database connection pools

MCS WAS components cluster organization



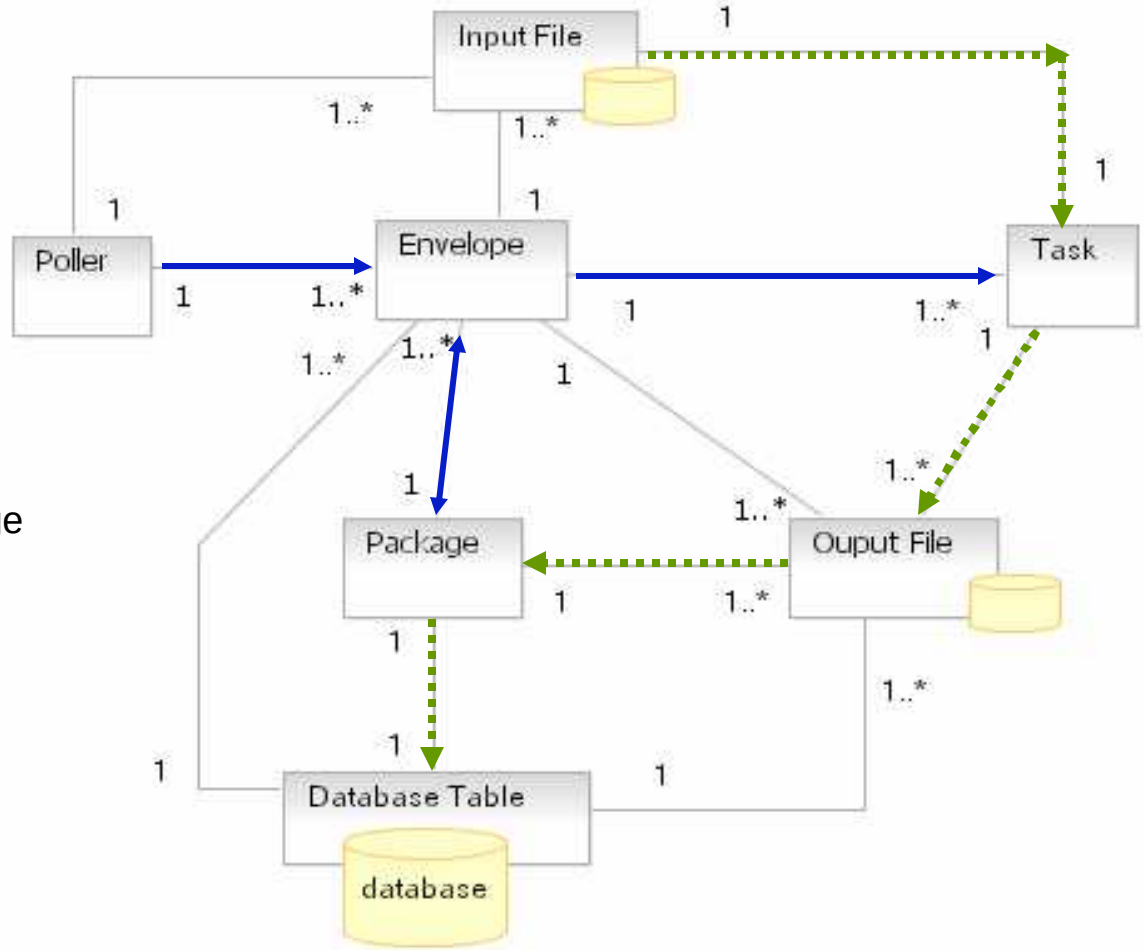
High-level application execution framework and component relations

- Components
  - State machine driven:
    - Poller
    - Envelope
    - Task
    - Package
  - Managed data
    - Input file
    - Output file
    - Database tables



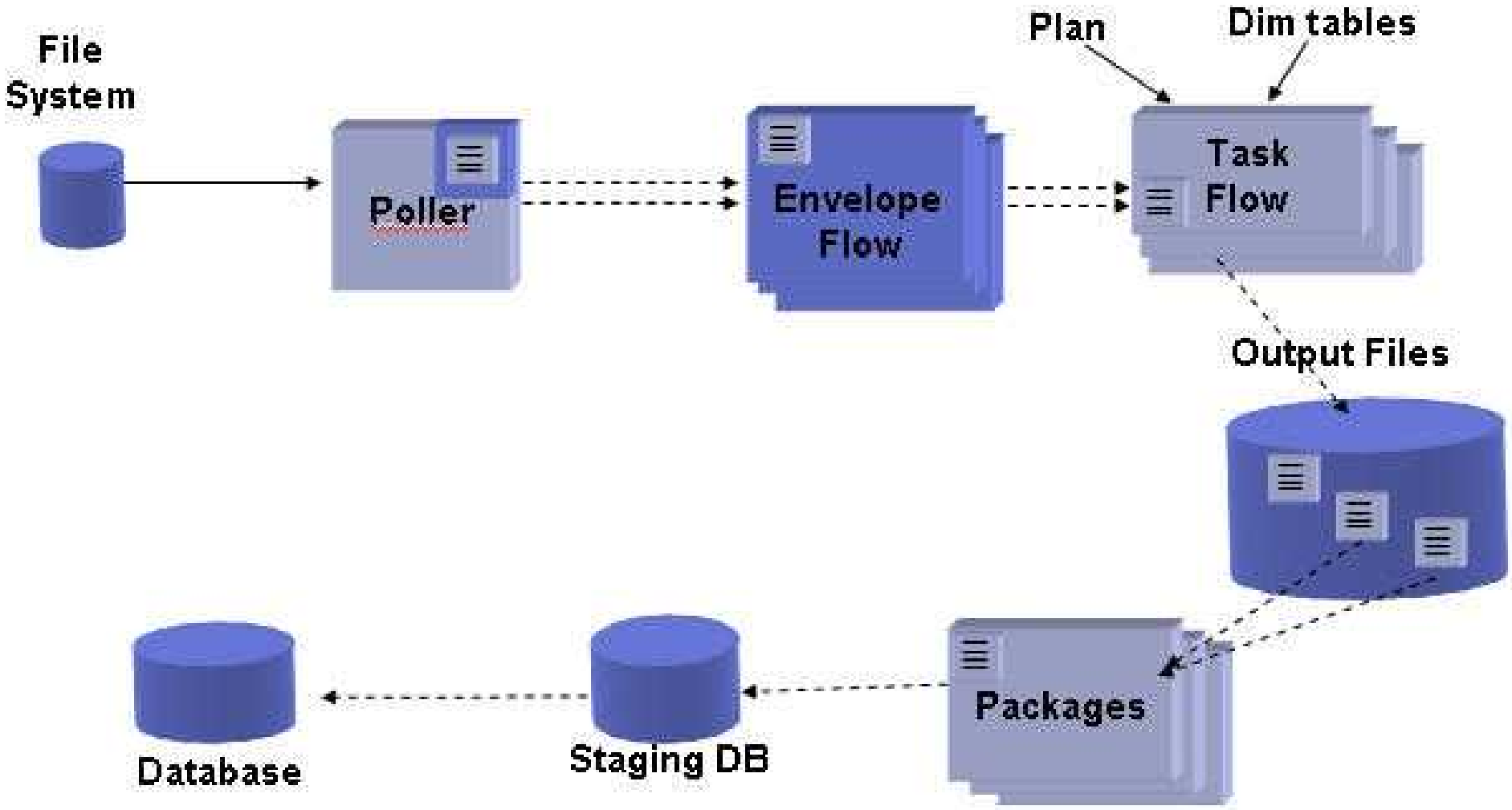
High-level components interactions

> Components
- State machine driven:
  - Poller
  - Envelope
  - Task
  - Package
- Managed data
  - Input file
  - Output file
  - Database tables

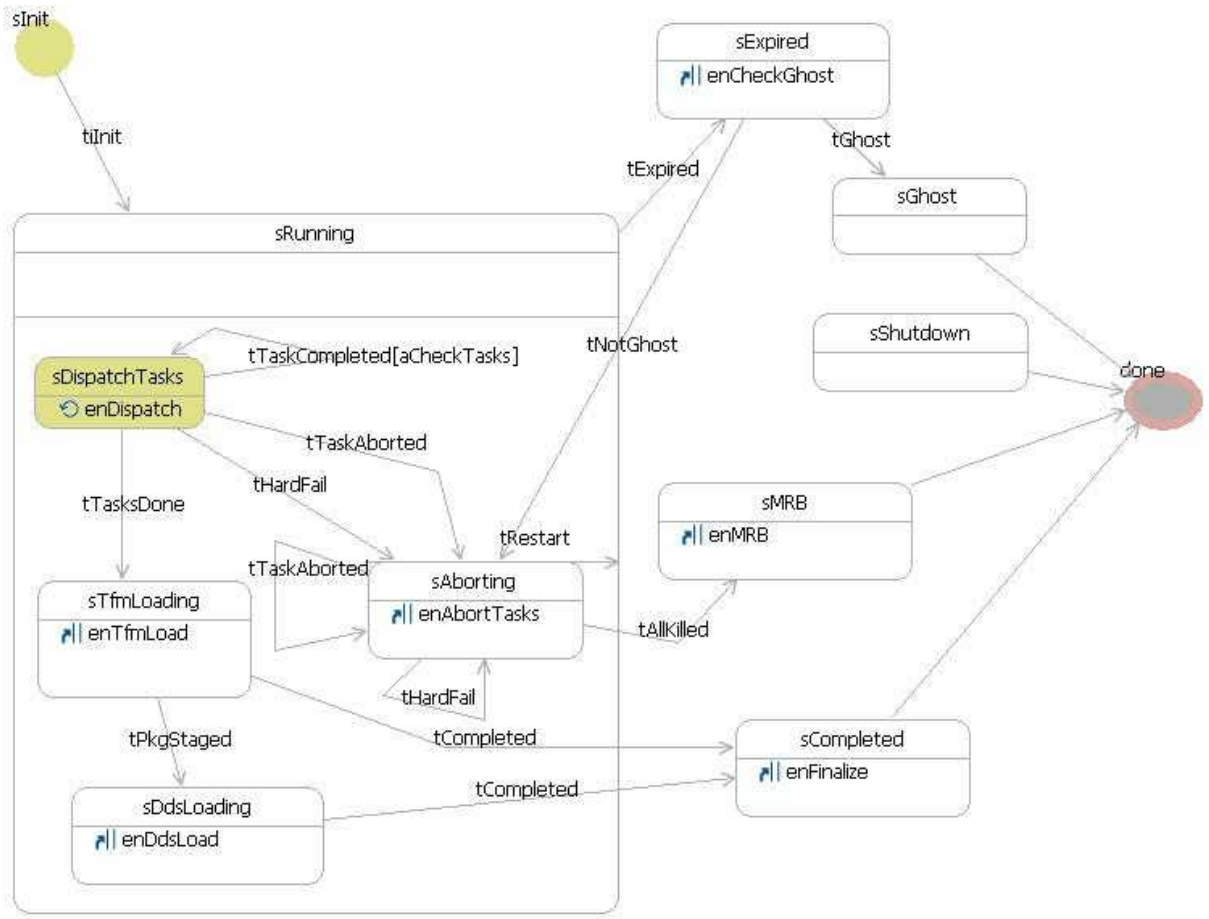


Asynchronous message invocations

Data flow



State Machine Diagram for the Envelope



MCS Data Objects (DOs)

- Facilitates persistency service for state machines and long lasting data objects.
- Long-lived state-machines
  - System singleton – the Poller
  - Configurable singletons – the Packages
- Short-lived dynamic state-machines
  - Tasks
  - Envelopes
- Long-lived dynamic data objects
  - Output files

DO table for dynamic, sort-lived state machines, or long-lived data objects

Id1	S1	St1	P1	…
Id2	S3	St1	P2	…
Id3	S1	St3	P3	…
Id4	S5	…	…	…
Id5	S2	…	…	…
id6	S5	…	…	…

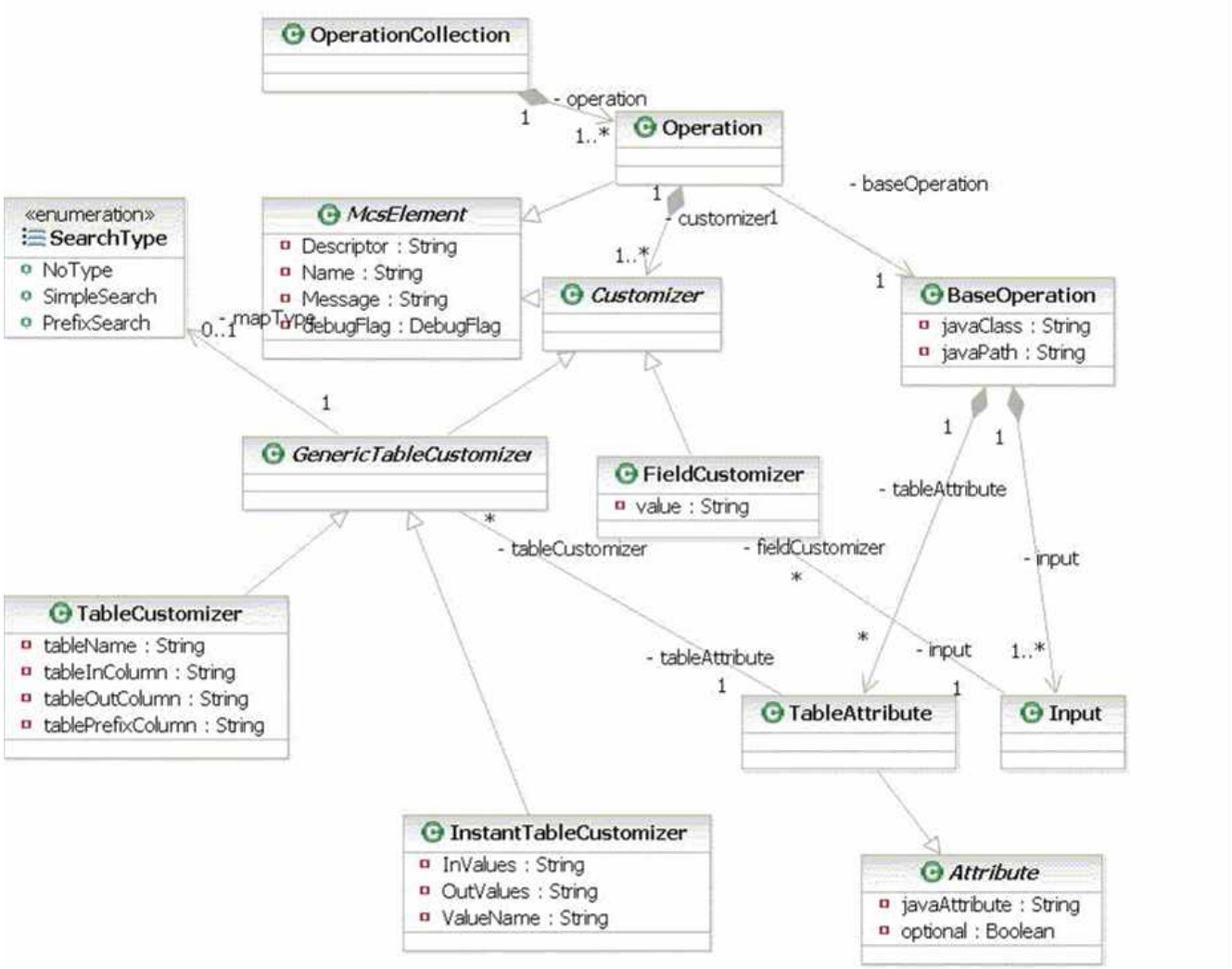
DO table for stastic, long-lived state-machines

Pl1	Sp1	Stp1	P1	…
Pl2	Sp3	Stp1	P2	…

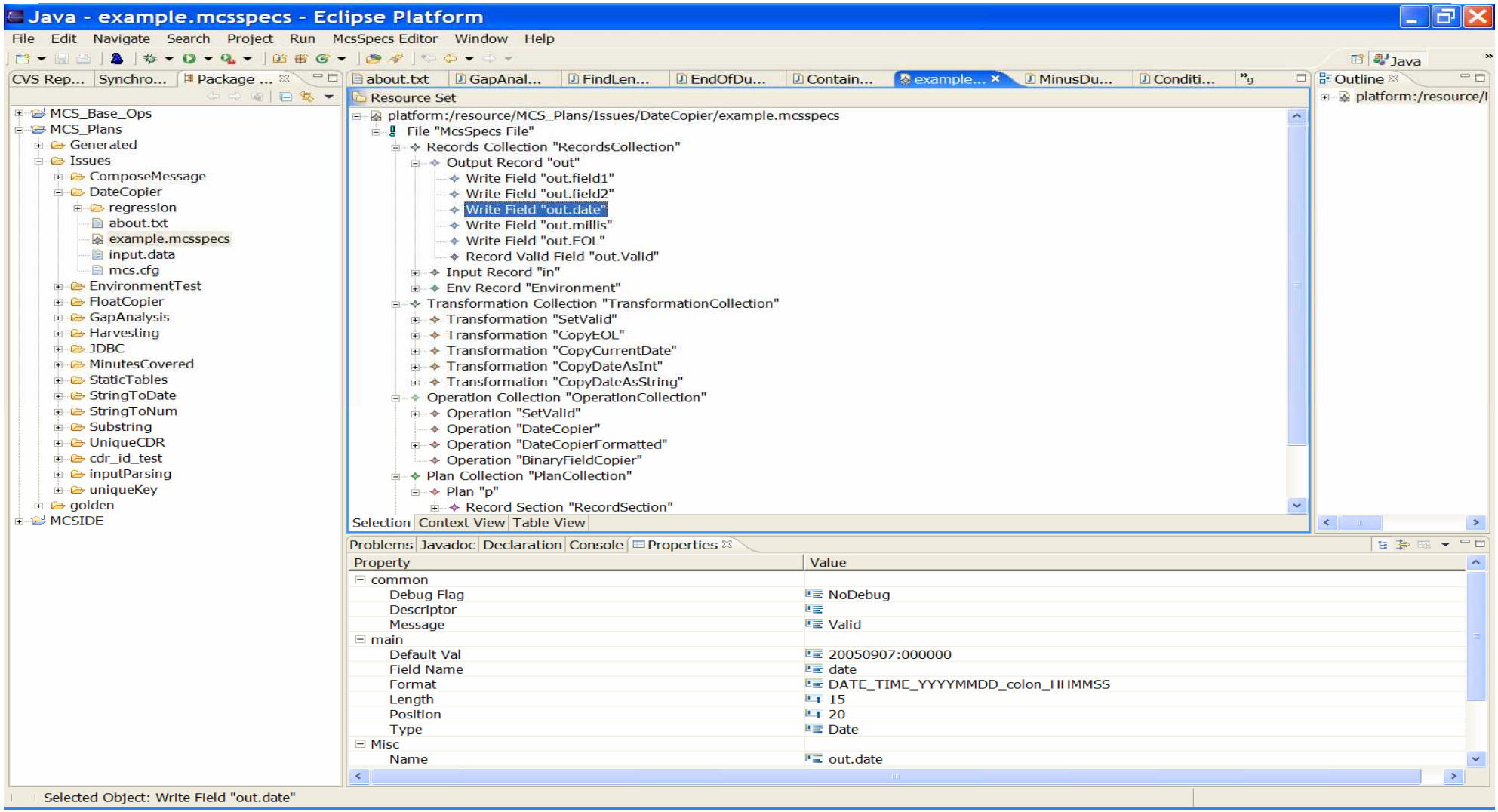
MCS Implementation Components – language side

- Domain-specific language execution environment
  - Pragmatic approach, algorithmic, using XML
    - No similarity to SQL or alike notations
  - Using (EMF) Eclipse Modeling Framework to model the language and parse its instances (i.e., programs)
  - Compiled to an efficient Java executable “compiland”
    - Reentrant and thread safe

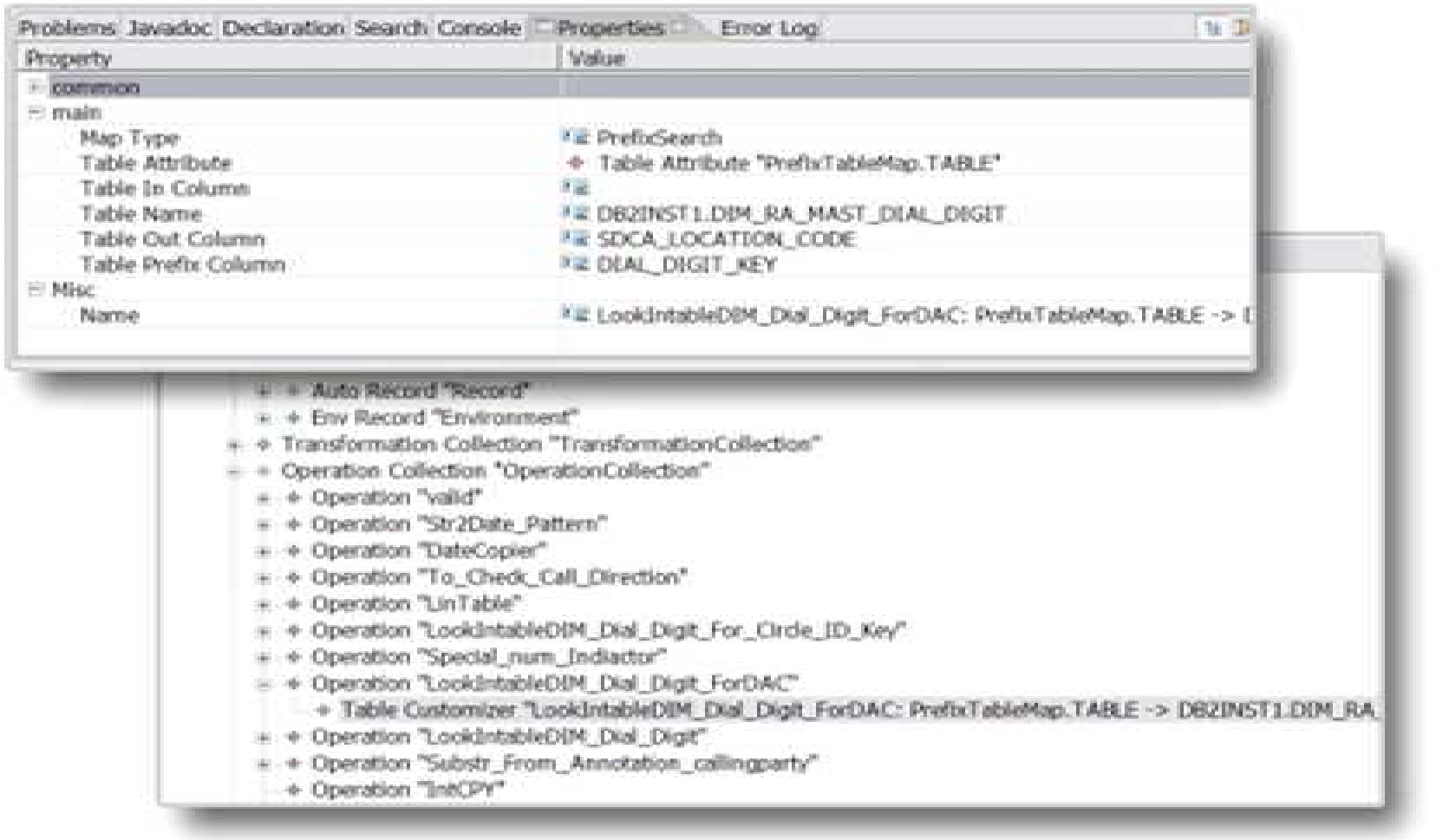
Language Modeling for EMF



MCS Language IDE Eclipse desktop



MCS Language IDE main windows: EMF Tree editor and the Properties view



Summary

- Problem in revenue assurance
- Collect and load data at very high rate and volume
- Solution based on:
  - Websphere Application Server for scalability, performance and enterprise-level solution
  - DB2 (DPF) partitioning and MDC
  - New Language for data transformation and processing
- Developed a high-level application framework on top of WAS
  - Using state-machine methodology
  - Hiding J2EE complexity
- Developed language specifications using EMF
  - IDE on Eclipse
  - Used by domain-experts
- Customer uses system throughout the development time
- Goals achieved.

Thank You