Rules Fest 2010

International Conference on Reasoning Technologies

October 11-14 • San Jose, CA USA

Grindwork Corporation
Intelligent Automation

JBoss® by Red Hat

IBM®

Production Systems Technologies

morris technical solutions
engineering knowledge for all businesses

VisionArts communications
strategic thinking • creative action

Rules-Based Applications: Design Decisions

George A. Williamson
Assoc. Systems Engineer
Decision Technologies
Union Pacific Railroad

Agenda

- Introduction
- Problem Analysis
- Implementation
- Testing
- Maintenance
- Complex Event Processing
- Use Case

Introduction

- George Williamson
- Assoc. Systems Engineer at Union Pacific Railroad
- Decision Technologies
- Internal Consultant for Rules-Based Development.
- A developer that builds software to satisfy real-world business needs.

Problem Analysis

What Role do Rules-Based Systems Play?

- **Business Adaptability**:
  Business rules are…
  - **Easy** to **modify**.
  - **Fast** to **deploy**.
- **Empower the Business Users**:
  Business users…
  - **Define** and **manage** the rules themselves.
  - **View**/**analyze**/**manage** rules directly through an **easy-to-use UI**. (rules ***are not*** embedded in code)
  - Provide **test scenarios** and **validate results**.
  - Control the **rule life-cycle**.

What Does "Adaptive" Mean?

- **Adaptive systems** provide **business users** the ability to **modify** and **enhance** their behavior ***quickly*** (daily/hourly), with **no code changes** nor the involvement of the IT department. This, thereby, **avoids delays** caused by project scheduling and development cycles.
- **Agile systems**, by contrast, enable ***code changes*** by ***IT developers*** to occur easily and quickly in a matter of ***months*** or ***weeks***, rather than quarters or years.
- Modern IT systems must be both: **Adaptive** ***AND*** **Agile**.

What is the Cost of Adaptability?

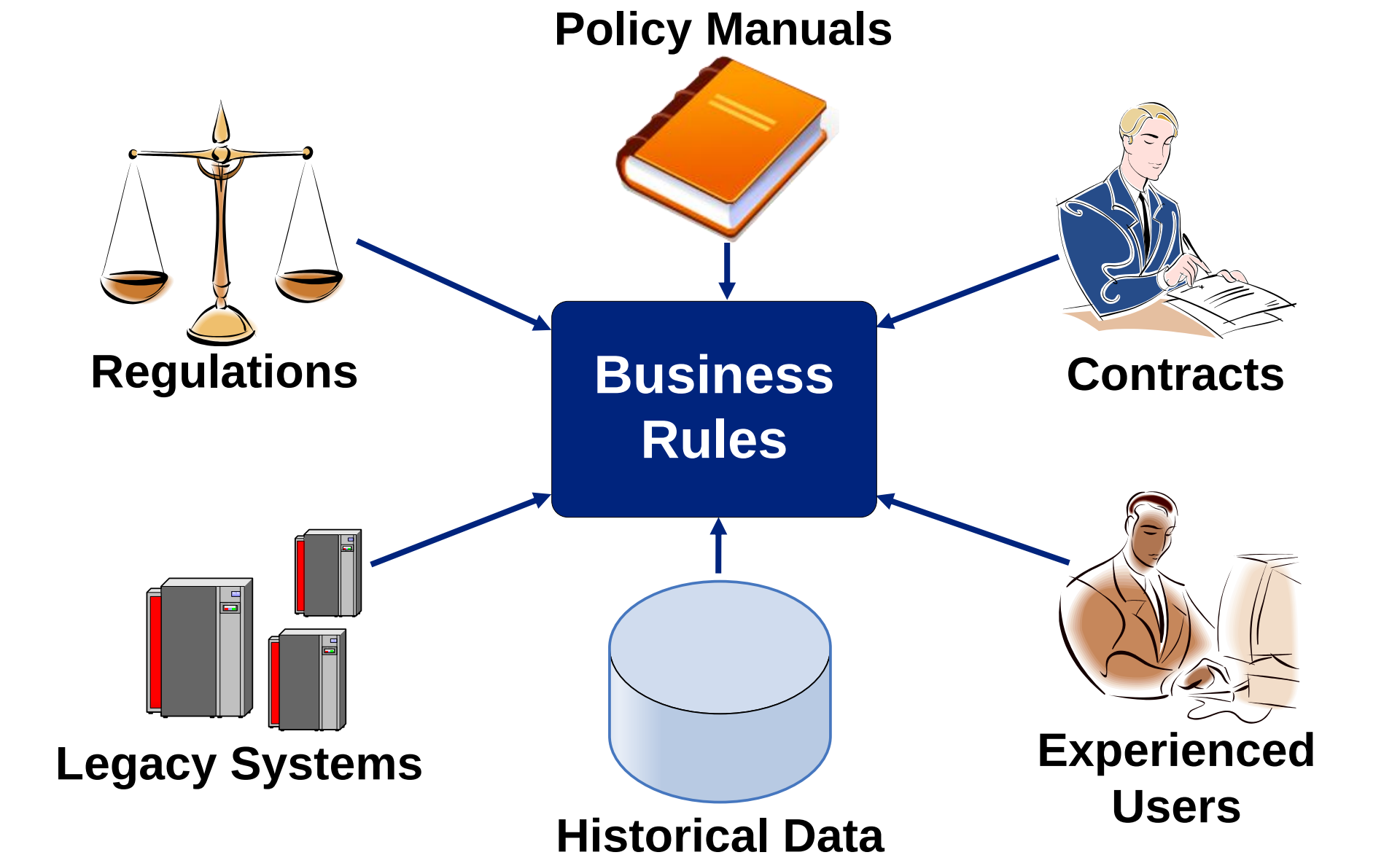
Adaptability isn't Free

- Merely adding rules-based technologies into a system doesn't automatically make it adaptable.
- Rules-based technologies provide a **framework** in which adaptable systems may be more easily built.
- Making the system adaptable requires:
  - More **analysis** in the design.
  - **Separation** between the **business rules** and the **implementation code**.
  - New **database tables**/**DAOs**.
  - Additional **user interfaces**.
  - More **integration points** with the rules engine.
  - Additional **development time** and up-front **cost**.

Do I Need a Rules Engine?

- Do the **benefits out-weigh** the **costs**?
- Is the problem **declarative**? The solution *does not* follow a **step-by-step procedure**.
- Are the business rules:
  - **Significant in number**?
  - **Sufficiently complex**?
  - **Changing frequently**?
- Can the business user **maintain** the rules? Is there a user that even **understands** all the business logic?
- Can the system be reasonably implemented in more **traditional technologies** (Java, et al.)?

Where are the Business Rules?



How Complex are the Rules?

Are the Rules Structured?

- **Structured rules** always follow a common pattern:
  - **Rules** all have the same **number of patterns**.
  - **Patterns** match on the same set of **business entities**.
  - **Constraints** operate on the same set of **attributes** using the same set of **predicate operators**.
  - All the rules can be abstracted to a **single logical expression** in which some clauses are unpopulated and, therefore, unconstrained.
  - **Similar actions** are taken as a result.
- **Decision Tables** are a "simple" spreadsheet-based representation of structured rules.

“Simple” Business Rules

- Rules (usually) operate on a **single business entity**.
- All rules follow a **common structure**.
- Constraints compare one **attribute** to one or more **literal values**: X = 1, X != 3.2, X starts-with “A”, etc.
- No constraints compare **attribute-to-attribute**: ~~X=Y~~
- **Disjunctive Normal Form** (**DNF**) – Disjunction (OR) of conjunctive (AND) clauses. Each **argument** is a **constraint** and each **clause** is a **rule**:
  $(C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n}) \vee (C_{21} \wedge C_{22} \wedge \ldots \wedge C_{2n}) \vee \ldots \vee (C_{m1} \wedge C_{m2} \wedge \ldots \wedge C_{mn})$
- **Spreadsheet**/**Decision Table** rule representation.
- **Non-technical users** can easily **maintain** the rules.

“Moderate” Business Rules

- Rules operate on **one** or **many business entities**.
- All rules follow a **common structure**.
- Constraints compare **attributes** to **literal values** *and/or* **other attributes**: X = 1, X != 3.2, X = Y, etc.
- Rules evaluate multiple attributes at once using **complex logical expressions**:
  (A_1 starts-with “A”) $\wedge$ ((A_2 is null) $\vee$ (A_2 equals A_3))
- **No Spreadsheet/Decision Table** representation may be sufficient for all rules.
- **Form-based** rule representation.
- **Non-technical users** can **maintain** the rules.

“Complex” Business Rules

- Rules usually operate on **many business entities**.
- Rules have **no common structure**; rules are distinct
- Constraints compare attributes to each other using **complex logical expressions**.
- **No simple**, **common** rule representation exists.
- An **IT developer** specializing in rules-based technologies **writes** and **maintains** the rules.
- Business users have **little visibility** into the encoded rules and are **unable to change** them.
- Used to automate **extremely complex behavior**.
- **Event Processing** rules often fall into this category.

Implementation

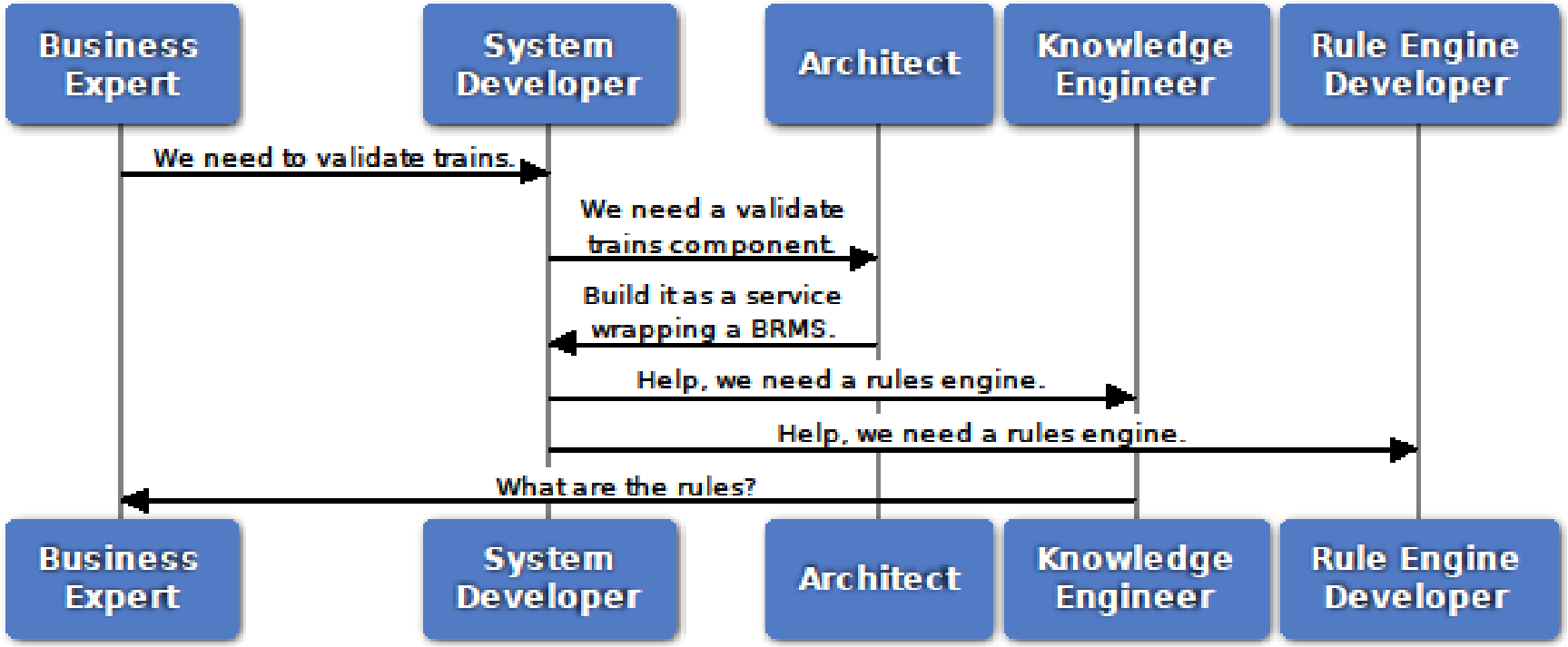
What Components Need to be Built?

- **Rules Engine**
  - **Accessible**
  - **Scalable**
  - **Available**
  - **Testable**
- **Rules Repository**
- **Domain-Specific Language**
- **Rules Management User Interface (Web-Based)**
- **Administration**

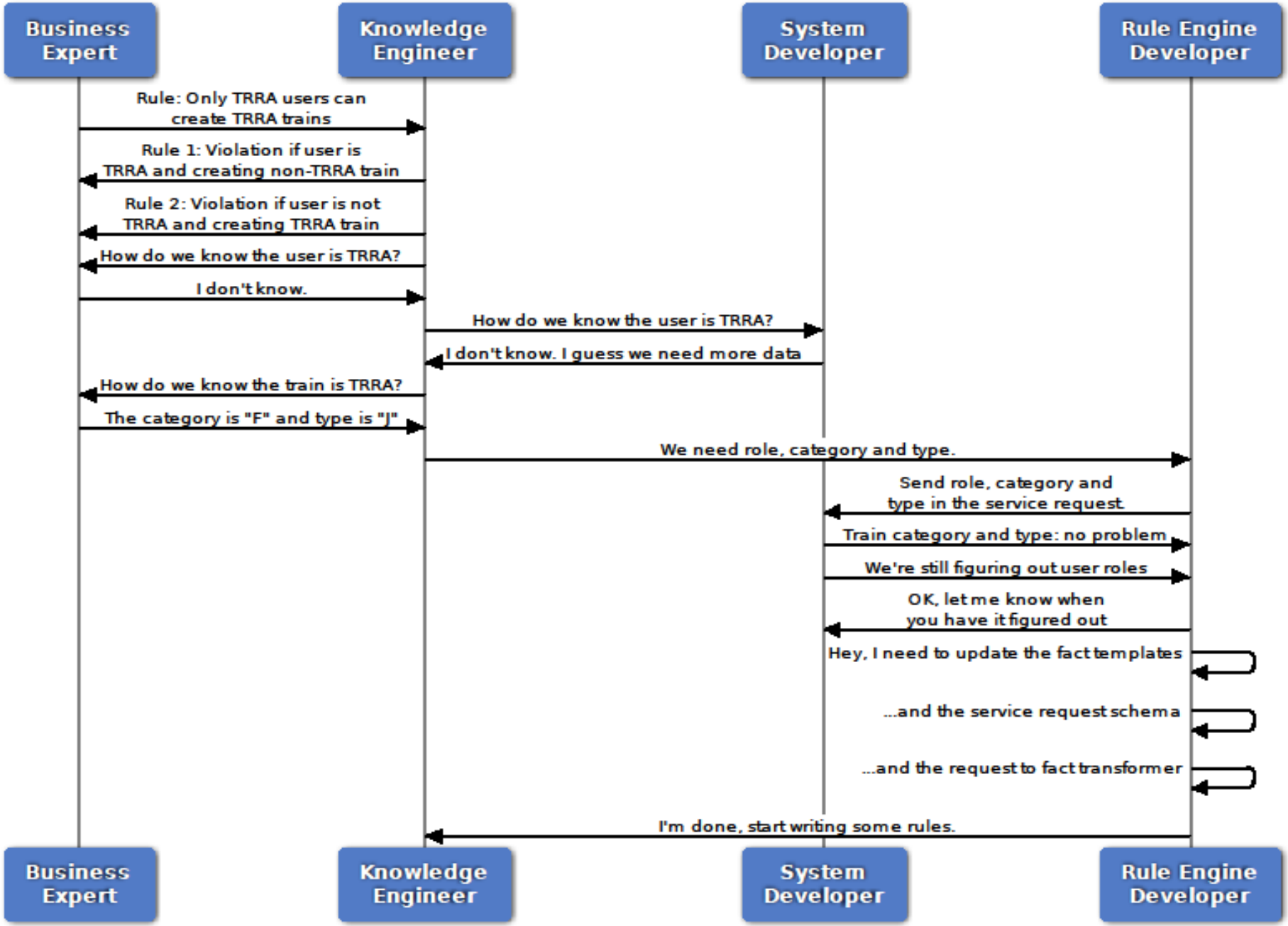
Who is going to build it?

- **Business Expert** – **non-technical** person outside of IT that knows the business and determines what the system should do. In an ideal world, they **encode** and **maintain** the **business rules**.
- **Knowledge Engineer** – Work with the business expert to **extract the rules** and **formalize the logic** to be encoded into a rules engine.
- **Rule Engine Developer** – **technical specialists** in rules-based **development** and **utilities**.
- **System Developer** – **conventional programmers** that build the rest of the IT system.
- **Architect** – **technical generalists** that determine how the system should best be structured.

What is the Typical Interaction?



What is the Typical Interaction?



Do I Need an Enterprise-Strength BRMS?

- **Enterprise BRMS Products** enable **faster development** and **easier maintenance**:
  - **Rule Representations/Abstractions**: Natural Language, Decision Tables, Decision Trees
  - **Rules Repositories**: reuse, versioning, source control
  - **Rules Management UI**: user-based maintenance
  - **Integration**: SOA, Database, Office Suite
  - **Deployment** & **Execution**:
    - **Runtime Administration**
    - **Scalability**: optimization, clustering, routing, memory caching
    - **Availability**: failover/failback, hot-deployment
  - **Testing**: completeness, ambiguity, simulation, regression
  - **Monitoring**: heart-beat, usage statistics, hit counts, JMX

How Much does an Enterprise BRMS Cost?

How Much does an Enterprise BRMS Cost?

- **Premium BRMS** products can be **very expensive**:
  > $1 million per enterprise license.
- **Specialized** rules-based products may be **less expensive** and appropriate, but **restrictive**.
- **Light-weight rules engines** are not as expensive.
  < $100,000 per enterprise license.
- **Open-source** BRMS products are getting better:
  - Domain-Specific Languages
  - Decision Tables
  - Rule Repositories
  - Rules Management UIs
  - Scalability/Availability
- More complex tools incur higher **training costs**.

Can I Get By with a Cheaper Product?

- Cheaper products usually have **fewer features**.
- This necessitates **custom implementations** of the required features that are missing from the product.
- In addition **price savings**:
  - **Easier integration** into the rest of the IT support structure.
  - Custom UIs often have **better usability**, making it easier for the business user to understand and maintain the rules.
  - **Scalability** and **availability** requirements are usually easy to achieve in **stateless**, **service-based** implementations by simply running multiple instances of the application.
  - Requires a **smaller learning-curve** on the part of the developer.
  - **System maintenance** is (in some ways) easier.

What is the Biggest Challenge?

- **Rules Management UI**
  - **Understandable** and **usable** by non-technical business experts. This depends greatly on the rule representation.
  - **Expressive**: the UI must be able to represent all the rules in the system, regardless of their complexity.
  - **Reusable** and **customizable** by multiple applications.
  - **Easily accessible**: web-based implementation.
  - **Testable**: The user must be able to see how a rule behaves before applying changes to the “production” environment.

Testing

How Do I Make Sure the Rules are “Good”?

- **Rules are maintained by business users and do not go through the typical IT change control process.**
- **Business rules spend their entire lifecycle in the “production” runtime environment.**
- **It is, therefore, very important to test the behavior of rules before they affect the business results.**
- **Many products supply rule analysis tools.**
- **Simulation Deployments are copies of the production system, often running on the same hardware, but they use a different set of rules and store their results for later examination without affecting the “production” business results.**

How Do I Test the Rules?

- **Simulation Deployments** are used for:
  - **Side-by-side Comparison** – Run a problem instance through both the production and simulation deployments and compare the results.
  - "**What-if** " **Validation** – Run all problem instances through both the production and simulation deployments. Capture the results of each environment and compare them once a sufficient amount of time has elapsed.
  - **Historical Comparison** – Sample problem instances are captured and stored with their expected results, computed from the current "production" system. When rules are changed and deployed to simulation, these sample problems are reprocessed and compared to the prior results. Changes are displayed to the user for validation and verification. **FIT** is very useful for implementing historical comparisons.

What is FIT?

- **Framework for Integrated Tests (FIT):**
  - **Users specify** what a program ***should*** **do.**
  - **Developers test** what it actually ***does*** **do.**
- Used throughout the software lifecycle:
  - **Users** specify system requirements by writing tests that depict the required behavior, before the code even exists.
  - **Programmers** execute the tests against the code they're writing to ensure the correct results are produced.
  - **Maintainers** verify that any modifications to the system either do not cause a test to fail, or work with the business user to change the test to reflect the system's new behavior.
- Many tools exist for FIT, most notably **FitNesse**, which utilizes **Wiki web pages**.

How do I use FIT to Test My Rules?

- FIT provides a convenient utility to implement **Regression Testing**.
- In addition to the input and expected output of the system, users may also specify the **business rules**.
- **Developers** write **light-weight** "**fixtures**" that:
  - **Transform** rules into the format required by the engine
  - **Add** the transformed rules to the engine.
  - **Run** the engine against the provided input.
  - **Test** that the expected output was produced.

Is FIT Only For Testing?

- **Business experts** and/or **knowledge engineers** can use this environment to:
  - **Capture** the **rules** that need to be encoded into the system.
  - **Structure** the **rules** into the appropriate representation.
  - **Identify** all the **entities** and **attributes** required for computation.
  - **Define** a **domain-specific language** (**DSL**) that makes the rules accessible to the business user.
  - **Early Prototyping** of the **rules management UI**.
  - **Collect test data** that may be used throughout the life of the system.

Maintenance

How Do I Make the System Maintainable?

- **Rules-based systems** are a **nightmare** to maintain, from an IT perspective.
  - Specialized **languages** and **development tools**.
  - Declarative programming **paradigm**.
  - **Impedance mismatch** between rules engine and traditional programming environment (Java).
  - Developers must be both **specialists** *and* **generalists**.
- **Business rules** are extremely **easy** to maintain.
  - Web-based rule **editors**.
  - **Domain-specific** languages.
  - **Spreadsheet** or **form-based** rule representations.
  - Rules **repositories**: versioning, staging, auditing, etc.

Maintenance: Enterprise vs. Light-Weight BRMS

- **Enterprise BRMS Products** provide solutions for many maintenance problems, but often cause more:
  - **Deployment models** don't match the company standard.
  - Often run in their own **containers**, which often require specialized **administration tools**.
  - Often use their own **monitoring tools**, which would need to be integrated into the company's monitoring system.
  - **Expensive**: annual licensing fee, which often increases.
  - **Training**: more features and specialized tools causes a greater learning curve for the maintainers of the system.
- **Light-weight BRMS Products** usually integrate easier into an existing IT infrastructure, but often lack the tools that make maintaining business rules easy.

What is an IT department to do?

- Dedicated, **maintenance groups** rarely have the skill set needed to maintain rules-based systems.
- **Centers of Expertise** – groups dedicated to **building** *and* **maintaining** rules-based systems.
- **Enterprise BRMS** products can help with some issues, but introduce more overhead than a lighter-weight product.
- **Rules Engine Frameworks** can provide a rubber-stamp pattern for implementing and deploying all rules-based applications within a company.

Complex Event Processing

Are CEP Systems Rules-Based Systems?

No!

(Well…, yes. Some of them kind of are.)

How is CEP NOT Rules-Based?

- **Rules Based Systems**:
  - **Automate** business decisions.
  - **Easily** and **quickly modify** the system's behavior as the business changes.
  - **Ensure** that the system makes the correct decisions.
- **Complex Event Processing**:
  - **Filter**, **Aggregate** and **correlate** business events as they occur, in **real-time**.
  - **Identify** situations-of-interest.
  - **Inform** those responsible for dealing with them.
  - **Act** on those situations automatically.
  - Achieve an **ultra-low latency**.

How is CEP Related to Rules-Based Systems?

- **Filtering**, **correlating**, and **aggregating** events is a **many-to-many pattern-matching** problem.
- The **Rete Algorithm**, which forms the core of most modern rules engines, is the fastest many-to-many pattern matching algorithm ever developed.
- In a **rules-based CEP system**:
  - **Incoming events** are transformed to **facts** and asserted into working memory on arrival.
  - **Rules** analyze the event to apply **filters** and **correlate** and/or **aggregate** it with prior events.
  - **New events** are sent (asserted) as a result of firing a rule.
  - **Working memory** provides a **stateful**, **current world view** that changes in **real-time** as events are processed.

What are the Requirements of Real-Time CEP?

- Real-time Event Processing places **memory** and **performance** requirements on the system that are not common for other application types:
  - Process **large volumes** of events (> 1 million/day).
  - Processing must be **fast**:
    - Each event must be processed in a **timely manner**.
    - **Database lookups** are **too slow**. All data required for processing must be in-memory when the event arrives (**delta-processing**).
    - A **shared**, **distributed working memory** is needed to support concurrent changes by different, heterogeneous processes.
    - **Multi-threading**/**multi-processing**: some tasks may be too complex (processing time*event volume) to be performed by a single process. This requires **synchronization** of working memory.
  - Events **cannot be lost**:
    - **JMS queues**/**topics** can **overflow**, resulting in lost messages.
    - **Working memory** must be **rebuilt quickly** after system failure.

Use Case

Car Cycle Event Sequencing

Use Case

- **Car Hire** is the amount one railroad pays to another for the use of their railcars.
- An important factor in computing Car Hire is the **actual number of miles** traveled by the car.
- This mileage is computed based on the **events** reported for that car as it traverses the rail network, particularly train **arrivals** and **departures**.
- For many different reasons, events may be reported in an **incorrect sequence**, resulting in an incorrect mileage computation.
- **Sequencing events** appropriately is, therefore, critical in calculating the correct car hire payment.

Basic Rules of Car Cycle Sequencing

Use Case

- A **departure** on a train should be followed by an **arrival** on the **same train**.
- An **arrival** at some location should be followed by a **departure** at the **same location**.
- **Non-movement events** should occur **between** the **arrival** and **departure** events at the same location.
- **On-line events** should occur at locations that are **between** the **previous departure** and **next arrival** locations.

A Well-Formatted Car Cycle

Use Case

```
RL 07/29/00 0830 HB004  DALLERUP        TX SL/L
FI 07/29/00 0835 HB004  DALLERUP        TX YBS70X 29
YD 07/29/00 1100 HB004  DALLERUP        TX YBS82  29
YA 07/29/00 1130 LS372  ENGLEWOOD       TX YBS82  29
YD 08/01/00 2345 LS372  ENGLEWOOD       TX YEW99W 31
YA 08/02/00 0015 B 372  HOUSTON         TX YEW99W 31
HD 08/02/00 0300 B 372  HOUSTON         TX
RH 08/05/00 1715 B 372  HOUSTON         TX
TD 08/05/00 2015 B 372  HOUSTON         TX MHOWCB 03
TA 08/07/00 0440 AX340  SAN ANTONIO     TX MHOWCB 03
TD 08/07/00 0622 AX340  SAN ANTONIO     TX MHOWCB 03
TA 08/07/00 1310 SO387  DEL RIO         TX MHOWCB 03
TD 08/07/00 1416 SO387  DEL RIO         TX MHOWCB 03
TA 08/08/00 1347 SO605  ALPINE          TX MHOWCB 03
TD 08/08/00 1351 SO605  ALPINE          TX MHOWCB 03
TA 08/09/00 1706 SP149  LORDSBURG       NM MHOWCB 03
TD 08/09/00 1721 SP149  LORDSBURG       NM MHOWCB 03
TA 08/10/00 1006 SP549  BLAISDELL       AZ MHOWCB 03
TD 08/10/00 1007 SP549  BLAISDELL       AZ LKT42  10
TA 08/10/00 1303 LB128  NOGALES         AZ LKT42  10
ID 08/10/00 1710 LB128  NOGALES         AZ FXE
```

Car Event Timestamp Issues

Use Case

- **Car events are initially ordered based on timestamp, i.e. the time at which the event was initially reported.**
- **Many systems across the railroad generate car events, each using their own internal time.**
- **These system's times are not necessarily synchronized, which causes events to be reported out of order.**
- **Some devices hold on to events for some amount of time before reporting them. Therefore, the reported time can be inaccurate.**

Cars Miss-Reported on a Train

Use Case

- **Occasionally, the system will think a car was placed on a train when in actuality it is still sitting in the yard.**
- **In this situation, events generated for the reported train will cause erroneous car cycle events to be reported for the car.**
- **As a result, the car cycle will depict a route the car never traveled.**

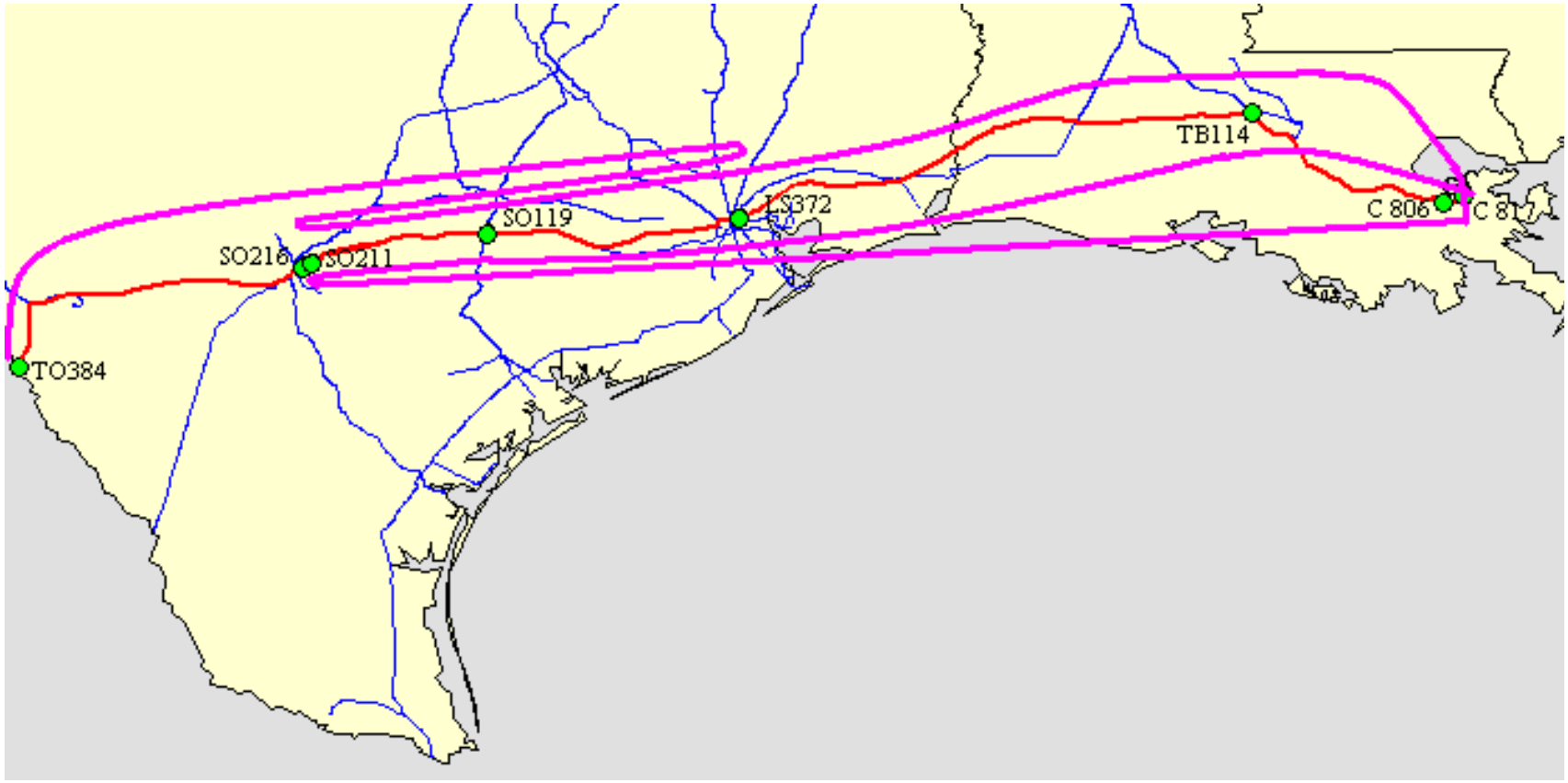
A Problematic Car Cycle

Use Case

```
RL 12/10/99 1918 TO384 EAGLE PASS          TX CU/E
FI 12/11/99 0730 TO384 EAGLE PASS          TX YEA77  11
TD 12/12/99 1230 TO384 EAGLE PASS          TX MEGEY  12
TA 12/12/99 1950 SO216 SAN ANTONIO EYARD   TX MEGEY  12
TD 12/13/99 1657 SO216 SAN ANTONIO EYARD   TX MWCEWB 07
TA 12/13/99 1718 LS372 ENGLEWOOD           TX MWCEWB 07
OT 12/13/99 1721 SO216 SAN ANTONIO EYARD   TX
TD 12/15/99 1423 LS372 ENGLEWOOD           TX MEWLI  22
TA 12/19/99 0600 TB114 LIVONIA             LA MEWLI  22
TD 12/19/99 0600 TB114 LIVONIA             LA MLIAV  03
TA 12/19/99 0600 C 806 AVONDALE            LA MLIAV  03
YD 12/19/99 0600 C 806 AVONDALE            LA YAV98  03
YA 12/19/99 0600 C 817 NEW ORLEANS         LA YAV98  03
YD 12/19/99 0600 SO216 SAN ANTONIO EYARD   TX YEY68  19
YA 12/19/99 0600 SO211 KIRBY               TX YEY68  19
YD 12/19/99 0600 SO211 KIRBY               TX YEY68  19
YA 12/19/99 0602 SO216 SAN ANTONIO EYARD   TX YEY68  19
TD 12/19/99 1038 SO216 SAN ANTONIO EYARD   TX MWCEWB 13
TA 12/19/99 1547 SO119 FLATONIA            TX MWCEWB 13
TD 12/19/99 1547 SO119 FLATONIA            TX MWCEWB 13
TA 12/21/99 0606 LS372 ENGLEWOOD           TX MWCEWB 13
ID 01/03/00 2245 C 817 NEW ORLEANS         LA IC
```

A Problematic Car Cycle

Use Case



A Problematic Car Cycle

Use Case

```
RL 12/10/99 1918 TO384 EAGLE PASS          TX CU/E
FI 12/11/99 0730 TO384 EAGLE PASS          TX YEA77  11
TD 12/12/99 1230 TO384 EAGLE PASS          TX MEGEY  12
TA 12/12/99 1950 SO216 SAN ANTONIO EYARD   TX MEGEY  12
TD 12/13/99 1657 SO216 SAN ANTONIO EYARD   TX MWCEWB 07
TA 12/13/99 1718 LS372 ENGLEWOOD          TX MWCEWB 07
OT 12/13/99 1721 SO216 SAN ANTONIO EYARD   TX
TD 12/15/99 1423 LS372 ENGLEWOOD          TX MEWLI  22
TA 12/19/99 0600 TB114 LIVONIA            LA MEWLI  22
TD 12/19/99 0600 TB114 LIVONIA            LA MLIAV  03
TA 12/19/99 0600 C 806 AVONDALE           LA MLIAV  03
YD 12/19/99 0600 C 806 AVONDALE           LA YAV98  03
YA 12/19/99 0600 C 817 NEW ORLEANS        LA YAV98  03
YD 12/19/99 0600 SO216 SAN ANTONIO EYARD   TX YEY68  19
YA 12/19/99 0600 SO211 KIRBY              TX YEY68  19
YD 12/19/99 0600 SO211 KIRBY              TX YEY68  19
YA 12/19/99 0602 SO216 SAN ANTONIO EYARD   TX YEY68  19
TD 12/19/99 1038 SO216 SAN ANTONIO EYARD   TX MWCEWB 13
TA 12/19/99 1547 SO119 FLATONIA           TX MWCEWB 13
TD 12/19/99 1547 SO119 FLATONIA           TX MWCEWB 13
TA 12/21/99 0606 LS372 ENGLEWOOD          TX MWCEWB 13
ID 01/03/00 2245 C 817 NEW ORLEANS        LA IC
```

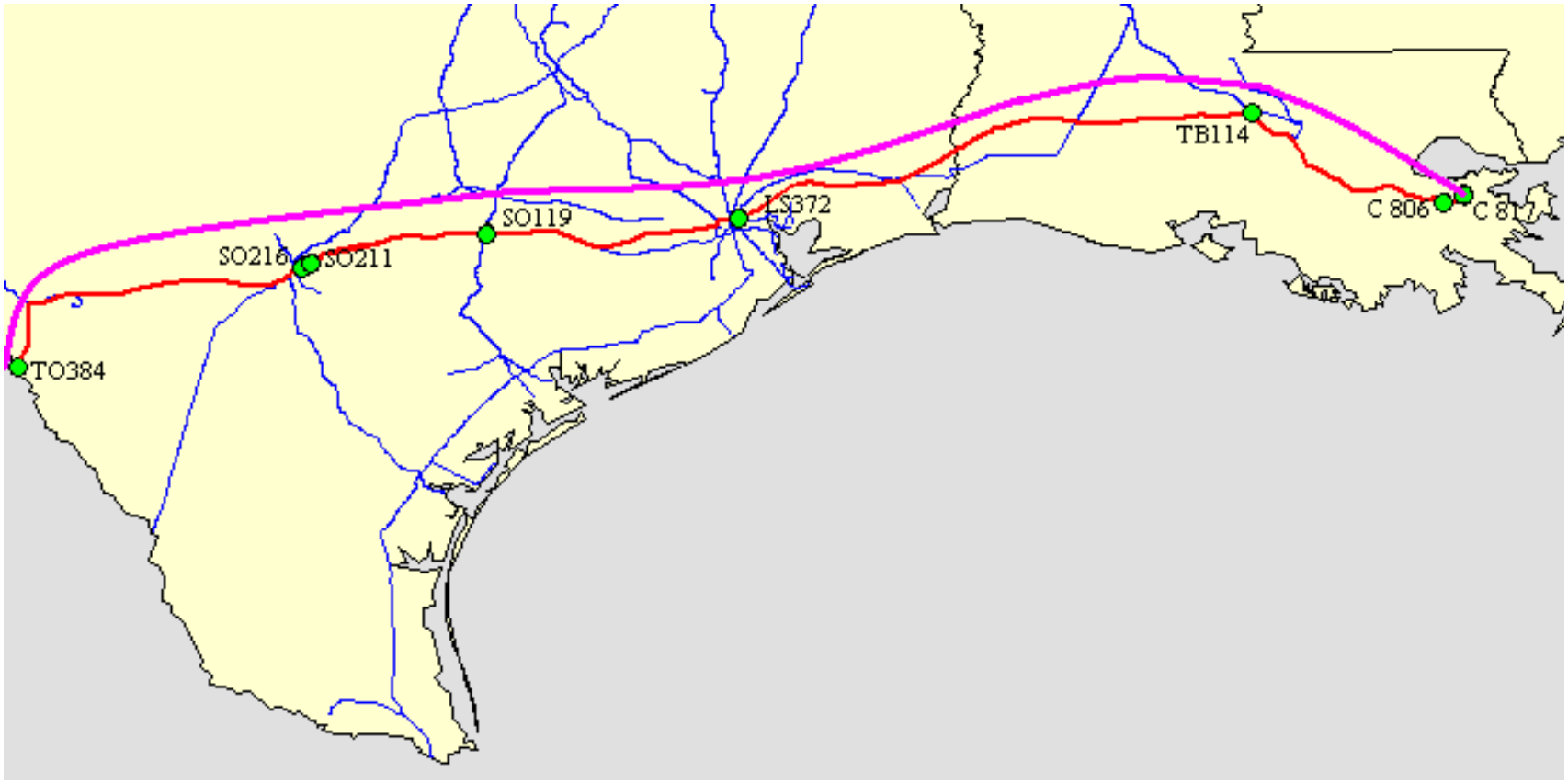
The Corrected Car Cycle

Use Case

```
RL 12/10/99 1918 TO384 EAGLE PASS          TX CU/E
FI 12/11/99 0730 TO384 EAGLE PASS          TX YEA77  11
TD 12/12/99 1230 TO384 EAGLE PASS          TX MEGEY  12
TA 12/12/99 1950 SO216 SAN ANTONIO EYARD   TX MEGEY  12
OT 12/13/99 1721 SO216 SAN ANTONIO EYARD   TX
YD 12/19/99 0600 SO216 SAN ANTONIO EYARD   TX YEY68  19
YA 12/19/99 0600 SO211 KIRBY               TX YEY68  19
YD 12/19/99 0600 SO211 KIRBY               TX YEY68  19
YA 12/19/99 0602 SO216 SAN ANTONIO EYARD   TX YEY68  19
TD 12/19/99 1038 SO216 SAN ANTONIO EYARD   TX MWCEWB 13
TA 12/19/99 1547 SO119 FLATONIA            TX MWCEWB 13
TD 12/19/99 1547 SO119 FLATONIA            TX MWCEWB 13
TA 12/21/99 0606 LS372 ENGLEWOOD           TX MWCEWB 13
TD 12/15/99 1423 LS372 ENGLEWOOD           TX MEWLI  22
TA 12/19/99 0600 TB114 LIVONIA             LA MEWLI  22
TD 12/19/99 0600 TB114 LIVONIA             LA MLIAV  03
TA 12/19/99 0600 C 806 AVONDALE            LA MLIAV  03
YD 12/19/99 0600 C 806 AVONDALE            LA YAV98  03
YA 12/19/99 0600 C 817 NEW ORLEANS         LA YAV98  03
ID 01/03/00 2245 C 817 NEW ORLEANS         LA IC
```

The Corrected Car Cycle

Use Case



Rules-Based Solution

Use Case

- A **rules-based application** that re-sequences car cycle events was implemented in 2001.
- **Design**:
  - Written in **C++** using the **CLIPS** rule engine utility.
  - **5 months** of development using **2.5 resources**.
  - **Complex**, hand-coded rules.
  - **Post processing**: run once after car cycle closes.
- **Costs/Savings**:
  - Total cost of development: < $**100,000**.
  - Greater Coverage: **15**% → **5**% threshold.
  - Car Hire savings benefit once delivered: ~$**2,500,000/mo**.
  - Savings in personnel: ~**25 FTE/year**.

Thank You,

Any Questions?