TIBCO BusinessEvents®

Architect's Guide

Software Release 5.6
August 2019

TIBC⦿®

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix, ActiveMatrix BusinessWorks, ActiveSpaces, TIBCO Administrator, TIBCO BusinessEvents, TIBCO Designer, Enterprise Message Service, TERR, TIBCO FTL, Hawk, TIBCO LiveView, TIBCO Runtime Agent, Rendezvous, and StreamBase are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Contents

Figures

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

TIBCO_HOME/release_notes/TIB_businessevents-enterprise_5.6.0_docinfo.html where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is C:\tibco. On UNIX systems, the default *TIBCO_HOME* is /opt/tibco.

The following documents for this product can be found in the TIBCO Documentation site:

- *TIBCO BusinessEvents Release Notes*
- *TIBCO BusinessEvents Installation*
- *TIBCO BusinessEvents Getting Started*
- *TIBCO BusinessEvents Architect's Guide*
- *TIBCO BusinessEvents Administration*
- *TIBCO BusinessEvents Developer's Guide*
- *TIBCO BusinessEvents Cloud Deployment Guide*
- *TIBCO BusinessEvents Data Modeling Developer's Guide*
- *TIBCO BusinessEvents Event Stream Processing Pattern Matcher Developer's Guide*
- *TIBCO BusinessEvents Event Stream Processing Query Developer's Guide*
- *TIBCO BusinessEvents Configuration Guide*
- *TIBCO BusinessEvents WebStudio User's Guide*
- *TIBCO BusinessEvents Decision Manager User's Guide*
- Online References:

    - *TIBCO BusinessEvents Java API Reference*
    - *TIBCO BusinessEvents Functions Reference*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit http://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

Complex Event Processing (CEP)

Complex Event Processing (CEP) is a set of technologies that allows *events* to be processed on a continuous basis.

Most conventional event processing software is used either for Business Process Management (BPM), TIBCO ActiveMatrix® BPM for example, or for Service Oriented Architecture (SOA), for example TIBCO ActiveMatrix® BusinessWorks software.

CEP is unlike conventional event processing technologies, however, in that it treats all events as potentially significant and records them asynchronously.

Applications that are appropriate for CEP are event-driven, which implies some aspect of real-time behavior. The typical CEP application area can be identified as having some aspect of "situation awareness," "sense and respond," or "track and trace" aspects which overlap in actual business situations.

Situation Awareness

Situation awareness is about "knowing" the state of the product, person, document, or entity of interest at any point in time. Achieving this knowledge requires continuous monitoring of events to do with the entity, events that indicate what situation or state the entity is in, or about to be in. As an example, a dashboard indicates all performance indicators for a runtime production process. All the production plant events are monitored and the situation, or health, of the production process is determined via some performance indicators that are shown in real-time to one or more operators.

Sense and Respond

This aspect is about detecting some significant fact about the product, person, document or entity of interest, and responding accordingly. To achieve this result the software does the following:

- Monitors events that indicate what is happening to this entity.

- Detects when something significant occurs.

- Executes the required response.

As an example, you may monitor cell phone or credit card usage, detect that a cell phone or credit card is being used consecutively at locations that are too far apart for real-time person-to-business transactions. Detection of such transactions indicates that an act of fraud is in progress. The system responds accordingly, denying the transactions, and invoking the necessary workflow to handle the situation as defined in standard procedures.

Track and trace

This aspect is about tracking the product, person, document or entity of interest over time and tracing pertinent facts like location, owner, or general status. An example would be tracking events from an RFID-enabled inventory control system where at any point in time you need to know how many widgets are in what location.

"Situation awareness," "sense and respond," and "track and trace" can all be classified as types of *activity monitoring*, for which the continuous evaluation of incoming events is suitable. For this reason, CEP is often described as a generalization of *Business Activity Monitoring (BAM)*, although the event processing task may be only indirectly related to business, as in the case of an engine monitoring application or process routing task.

Technical Requirements of a CEP System

CEP systems must be able to receive and record events and identify patterns of these events and any related data. CEP systems must also handle temporal or time-based constraints, especially for handling the non-occurrence of events.

The following TIBCO BusinessEvents features satisfy these requirements:

- A rich event model, incorporating event channels (for different event mechanisms, such as different types of messaging software) and destinations (for different types of events).

- A pattern detection mechanism using a sophisticated, high performance, declarative rule engine.

- A backing store for historical depth, and to enable use of more event data and entity data that can be persisted using memory cache technologies.

The following add-on products enrich the functionality:

- A state model mechanism that allows entities to be described in terms of state and modelling of time-out events to handle the non-occurrence of events. (State modeling requires TIBCO BusinessEvents Data Modeling, purchased separately.)

- Query features that enable retrieval of specific data from the data store or from the event stream as it arrives, using SQL-like object query language.

- Pattern matching features that enable you to, for example, specify and identify the temporal order of event arrivals, and to correlate events across different event streams.

- The ability to import and use entity data that is stored in various enterprise data stores.

A Model-Driven Approach

The TIBCO BusinessEvents engine can be described not only as a CEP engine but also as an event-driven rule engine or real-time rule engine. TIBCO BusinessEvents enables CEP problems to be solved through a model-driven approach, in which the developer defines the event, rule, concept (class) and state models which are then compiled so that at run-time incoming events are processed as efficiently as possible.

The various models are as follows:

Event model

The event model describes the inputs into TIBCO BusinessEvents. Events provide information through their properties and (optionally) through an XML payload.

The event model provides the primary interface between TIBCO BusinessEvents and the outside world, for input as well as output. Typical event sources (or channels) are messages from TIBCO Rendezvous and TIBCO Enterprise Message Service middleware, events generated explicitly by TIBCO BusinessEvents, and other sources such as SOAP messages. Events can be used to trigger rules.

Concept model

The concept model describes the data concepts used in TIBCO BusinessEvents, which may be mapped from events or their payloads, or loaded by some other mechanism into TIBCO BusinessEvents.

The concept model is based on standard UML Class and Class Diagram principles.

Rule model

Rules provide one of the main behavioral mechanisms in TIBCO BusinessEvents. Rules are defined in terms of declarations (events and concepts of interest), conditions (filters and joins on and between the attributes of the events and concepts), and actions.

The underlying rule engine is based on an algorithm called the Rete algorithm, which mixes all rules together into a type of look-up tree, so that any additional concept instance or event can near-instantly cause the appropriate rule or rules to fire and invoke the appropriate actions. Rules are almost always defined in general terms (concepts or classes and events), so that they can apply to as many combinations of those events and classes that exist in memory at any one time. The combination of rule declaration and condition defines the event pattern required for CEP operation. Rule actions that update other concepts may cause other rules to become available for firing, a process called inferencing or forward chaining. These types of rules are generally called Production Rules.

Rule functions

Algorithms, procedures or functions may be usefully defined as parameterized rule functions and re-used as required in rule actions and other areas where a behavior can be specified.

State model

An important item of metadata for a concept or object is its state. Typically a state model describes the states that an entity can hold, and the transitions between states that are allowed, and the conditions for such transitions. Internally the state model is just additional metadata, but it is more useful to describe the state model as a visual model of states and transitions.

The state transition rules can be viewed as special customizations of standard rules. The state model is based on the standard UML State Model principles and requires TIBCO BusinessEvents Data Modeling add-on software.

Query model

Queries can provide both snapshot and continuous views of the data in a TIBCO BusinessEvents cache.

Queries can also provide continuous views of data arriving through channels. They are constructed and executed from rule functions in a specialized agent (called a query agent). Queries provide event stream processing or set operations to derive information that can then be used in rule functions, or shared (via events or the cache). This model requires TIBCO BusinessEvents Event Stream Processing add-on software.

Stateful Rule Engine

At run-time, the rule engine executes rules based on new events and data sources on a continuous basis.

The rule memory is never "reset" (unless by design), so that future events can always be compared to past events. For this reason, the rule engine is described as a *stateful rule engine*.

If required, the working memory can be cleared and a new set of data asserted for each "transaction," in which case the engine is operating as a *stateless rule engine.*

Object Management Types

TIBCO BusinessEvents offers an In Memory object management type, but for most use cases, persistence of data is required.

To provide TIBCO BusinessEvents with its enterprise and *extreme transaction processing* capabilities and to ensure resilience, TIBCO BusinessEvents provides a high performance distributed cache. The cache allows data to be persisted in memory and removed from the Rete network or returned to the Rete network, as required, to handle extremely large problem domains (domains that would not typically fit into a runtime memory model). A backing store can be added to provide additional reliability, and other functionality. Just as data can be moved between the Rete network and the cache, so can less used data be moved between the backing store and the cache, to balance storage, memory, and performance requirements.

Note that no rule operations are persisted. It is more efficient to simply rerun the rules and recreate the appropriate actions, than it is to persist the internal workings of the rule engine.

You can use the built-in cache provider, TIBCO BusinessEvents DataGrid, or integrate a supported version of Oracle Coherence, for which you have a license that is appropriate for your usage.

Main Product Components and Add-Ons

TIBCO BusinessEvents is a declarative, distributed event processing platform covering multiple event processing use cases.

Different use cases are supported using optional add-on products, available separately.

Main Components and Add-Ons
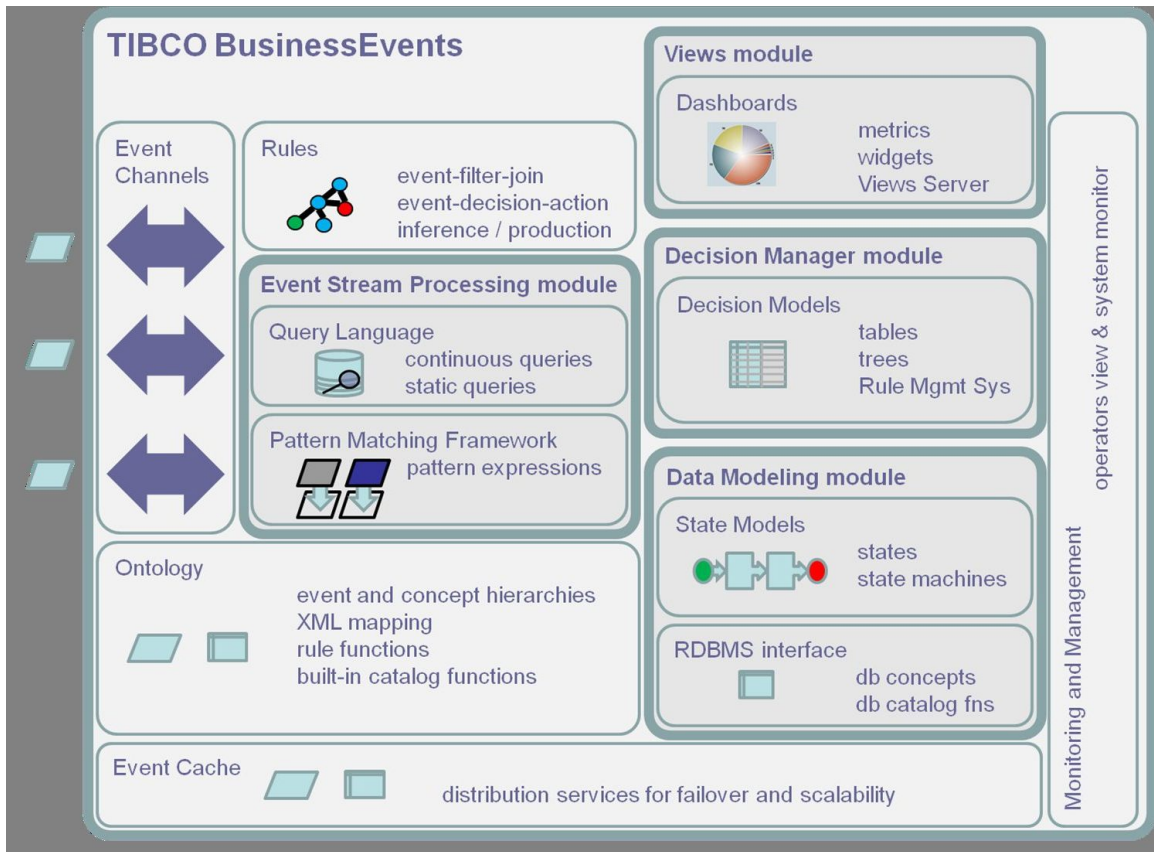
Add-on Products

The core product provides essential features such as channels, events and concepts, rules and rule functions, distributed cache, monitoring and management, tester, and so on.

Add-on products, available separately, support specific functions and roles. For example:

- A BPM user might be primarily interested in TIBCO BusinessEvents® Process Orchestration, with its ability to *segregate* different event processing rule sets within the flow of a BPM process

- TIBCO BusinessEvents® Decision Manager enables business analysts to construct detailed business rules using decision tables to represent actionable business rules, and a rules management server for workflow management.

- A BAM project might only need TIBCO BusinessEvents® Views dashboards, and supporting rules. TIBCO BusinessEvents Views provides real-time web-based dashboards that give visibility into the data flowing through a running TIBCO BusinessEvents application, using meaningful metrics that are presented to business users for proactive decision making.

- A sense and respond application might only use TIBCO BusinessEvents® Data Modeling, with supporting rules. TIBCO BusinessEvents Data Modeling provides entity lifecycle management using state machines, and direct database interaction using JDBC database concepts.

- A monitoring application could use TIBCO BusinessEvents® Event Stream Processing, and associated rules, possibly in conjunction with TIBCO BusinessEvents Data Modeling. TIBCO BusinessEvents Event Stream Processing provides continuous and snapshot queries and an event pattern matching framework.

TIBCO BusinessEvents Design-time Components

Design-time activities performed using the TIBCO BusinessEvents resources include building an ontology — a set of concepts, scorecards and events that represent the objects and activities in your

business — and building rules that are triggered when instances of ontology objects that fulfill certain criteria arrive in events.

The output of the design-time activities is an enterprise archive (EAR) file, ready to deploy (or configure for deployment as needed).

See tutorials in *TIBCO BusinessEvents Getting Started* to learn more.

- TIBCO BusinessEvents Studio: this is an Eclipse-based project building environment. It organizes project resources and makes the project organization and the project resources visible in graphical ways.

  Perspectives: The Eclipse plug-ins for TIBCO BusinessEvents and for TIBCO BusinessEvents add-ons provide these perspectives:

  TIBCO BusinessEvents Studio Development

  Provides resources for building TIBCO BusinessEvents projects.

  TIBCO BusinessEvents Studio Debug

  Provides resources for debugging rules and rule functions in TIBCO BusinessEvents projects, as well as testing running engines without debugging.

  TIBCO BusinessEvents Studio Diagram

  Provides interactive graphical views of a project that allow you to see relationships between project resources.

  TIBCO BusinessEvents Studio Decision Table

  Provides resources for building decision tables. (Available with TIBCO BusinessEvents Decision Manager.)

  TIBCO BusinessEvents Studio State Modeler

  Provides resources for building state models. It allows you to model states of ontology concept instances and use those states in rules. (Available with TIBCO BusinessEvents Data Modeling.)

- Integration with TIBCO ActiveMatrix BusinessWorks: TIBCO BusinessEvents communicates with TIBCO ActiveMatrix BusinessWorks through a provided plug-in that contains a palette of ActiveMatrix BusinessWorks Activities. Details are provided in TIBCO BusinessEvents Developer's Guide.

TIBCO BusinessEvents Administration Components

Administration of a deployed system involves management of objects generated by the inference engine, deploy-time configuration for tuning and other aspects of the system, deployment, management, and monitoring.

This section describes the TIBCO BusinessEvents components. For cache, you can optionally use a supported version of Oracle Coherence, for which you have a license that is appropriate for your usage. For deployment, monitoring, and management, you can optionally use TIBCO Administrator. Customers who are already using these software products may find it convenient to continue to do so. These products are not provided with TIBCO BusinessEvents.

Object Management

How you manage objects generated by the rules executing in the inference engine depends on whether you want to keep them for later use. You can manage objects in memory only, or using a distributed cache, or using a cache with a backing store. The recommended way to manage objects for most production needs is to use a cache and a backing store. When cache-based object management is used, agents of different types co-operate to provide efficient object storage and access, with options to use load balancing and fault tolerance of data and engine processes.

Object management is partly a design-time and partly an administration topic, because your choice of object management type can affect how you design rules. For example, you may have to retrieve

objects if they are stored only in the cache or only in the backing store, so they can be used in the Rete network. See Object Management Types for an introduction to these topics.

Deploy-time Configuration Settings are in the Cluster Deployment Descriptor (CDD)

Using the Cluster Deployment Descriptor (CDD) editor, you edit the CDD file to specify all the deploy-time properties for the entire cluster, from cluster-wide settings dealing with object management, through *processing unit* settings (that is, those at the TIBCO BusinessEvents engine level), to individual *agent class* settings.

To deploy any engine (processing unit) in the cluster, the only details needed are these: the EAR file, which contains all the project resources, the CDD file, and the name of the processing unit (a unit that deploys as an engine). You can change deploy-time configuration settings in the CDD file, without having to rebuild the EAR file.

TIBCO BusinessEvents Monitoring and Management (MM)

The MM component enables you to deploy cache-based or in-memory TIBCO BusinessEvents engines, and then monitor and manage a deployed cluster. It uses a canvas-based site topology editor to configure the physical deployment of the cluster. It also provides a web-based dashboard, the MM Console, to enable you to monitor the deployment and perform the various tasks.

You can configure the health metric and alert thresholds that define the graphical display of system health, and the actions to take when thresholds are reached, such as sending email. MM monitoring features enable you to easily spot bottlenecks or other trouble spots in the system so you can address any issues. MM also has a profiler and can generate other helpful reports

Deployment Topology Configuration Using a Site Topology Editor

See Deploy-time and Runtime Overview for details.

Design-time Resource Overview

In a rule engine, the things that the project works with such as employees, inventory, parts, and so on are *concepts* in the ontology of the project, as are *scorecards*, which hold metrics.

When TIBCO BusinessEvents Data Modeling software is used, a database concept feature enables you to create concepts from database data, and a state modeler feature enables you to model the behavior of concepts given certain occurrences.

Events such as flight take-off, purchase of a mortgage, sale of stock, and so on are also part of the ontology. Events can be created from messages arriving through channels. Events can also be generated internally, for use in the engine and to send out messages to external systems.

Rules are triggered by events and by changes in concepts and scorecards. For example, rules might cause baggage to be rerouted if there is a certain problem at the airport. Rule functions are functions written in the rule language that can be called from rules or other rule functions. Some rule functions serve special purposes at startup, shutdown, and in preprocessing events. When TIBCO BusinessEvents Data Modeling software is used, its *decision tables* also provide rules. These, however, are *business rules*, and are triggered only indirectly by the inferencing engine.

When TIBCO BusinessEvents Event Stream Processing software is used, you can design complex *queries* that provide information on the event stream or on cached objects that can in turn be fed into rules. You can also design event patterns to watch for, and take certain actions when they occur or do not occur.

Designing the ontology and the rules well is key to a good CEP (complex event processing) project.

Channels and Events

Channels (except for local channels which communicate between agents), represent connections to a resource, such as a Rendezvous daemon, JMS server, HTTP server or client, Hawk domain, or a space in TIBCO ActiveSpaces.

A channel has one or more *destinations*, which represent listeners to messages from that resource. Destinations can also be used to send messages to the resource.

Messages arriving through channels are transformed into *simple events*. Conversely, simple events sent out of TIBCO BusinessEvents are transformed to the appropriate type of message.

TIBCO BusinessEvents processes three kinds of events:

Simple Event

A representation of a single activity (usually a business activity) that occurred at a single point in time. Only simple events are used in channels.

Time Event

A timer. Generally created and used to trigger rules.

Advisory Event

A notice generated by TIBCO BusinessEvents to report an activity in the engine, for example, an exception.

TIBCO BusinessEvents creates instances of simple events and time events based on user-configured event definitions.

Concepts

A *concept definition* is a definition of a set of properties that represent the data fields of an entity.

Concepts are equivalent to UML Classes: they represent class-level information, and at runtime the instances of concepts are called objects.

Concepts can describe relationships among themselves. For example, an `order` concept might have a parent/child relationship with an `item` concept. A `department` concept might be related to a `purchase_requisition` concept based on the shared property, `department_id`.

With TIBCO BusinessEvents Data Modeling (purchased separately), concepts can include a state model. Also with TIBCO BusinessEvents Data Modeling, you can create concepts by importing table and view data from databases, and you can update the database definitions using RDBMS functions. These concepts are called *database concepts*.

Concept properties can be updated by rules and rule functions (including rule functions whose implementation is provided by decision tables).

Score Cards

A *score card* serves as a static variable that is available throughout the project. You can use a `ScoreCard` resource to track key performance indicators or any other information.

Use rules to view a scorecard value, use its value, or change its value. Note that unlike concepts and event definitions, which describe types of instances, each scorecard is both the description and the instance.

A score card is similar to a global variable, except that with multiple active inference agents, the value is local to the agent, and you can update the value of a scorecard in rules, but not the value of a global variable.

Rules

Rules define what constitutes unusual, suspicious, problematic, or advantageous activity within your enterprise applications.

Rules also determine what TIBCO BusinessEvents does when it discovers these types of activities. You can execute actions based on certain conditions which are defined using simple events, concept instances, events, score cards, or a combination of these objects.

TIBCO BusinessEvents offers the following types of functions for use in rules:

Standard Function

These functions are provided with TIBCO BusinessEvents.

Standard functions include a set of temporal functions, which allow you to perform calculations based on a sampling of a property's values over time. These functions make use of the history for that property.

Ontology Function

TIBCO BusinessEvents generates these functions based on the resources in your project.

Custom Function

You can write custom functions using Java and integrate them into TIBCO BusinessEvents for use in rules.

Rule Function

In addition to Java-based custom functions, you can use rule function resources to write functions using the TIBCO BusinessEvents rule language.

Object Management and Fault Tolerance

An important aspect of most TIBCO BusinessEvents applications is management of the objects created and modified at runtime.

It is important to consider the effect of object storage options when designing projects.

Different projects have different object management requirements. For some, it is acceptable to destroy the objects once the rule engine cycle that needs them has completed. They require only memory-based object management. For others, the instances have longer term value and need to be persisted.

Fault tolerance options are related to the object management type used. TIBCO BusinessEvents supports a variety of object management and fault tolerance options.

State Modeler

The state modeler feature is available only with the TIBCO BusinessEvents Data Modeling add-on software. State Modeler is based on the UML-standard definition for state models. It allows you to model the life cycle of a concept instance — that is, for each instance of a given concept, you can define which states the instance will pass through and how it will transition from state to state.

States have entry actions, exit actions, and conditions, providing precise control over the behavior of TIBCO BusinessEvents. Transitions between states also may have rules. Multiple types of states and transitions maximize the versatility and power of State Modeler.

See *TIBCO BusinessEvents Data Modeling Developer's Guide*.

Database Concepts

The database concepts feature is available only with the TIBCO BusinessEvents Data Modeling add-on software.

Database concepts are TIBCO BusinessEvents concepts with database behavior. They are created using a utility that enables you to map tables or views from a database to TIBCO BusinessEvents concepts.

See *TIBCO BusinessEvents Data Modeling Developer's Guide.*

Query Language and Framework

The query language and framework are available with TIBCO BusinessEvents Event Stream Processing add-on software.

The query language and framework enable you to perform set operations against cached concepts as well as against incoming event streams. Queries that obtain information at a point in time are called snapshot queries, and are available for cache queries only. Queries that listen to a message stream and collect information continuously are known as continuous queries.

Queries use an object-oriented SQL-like query language within rule functions. Query results can then be passed using events, or can be shared in cached concepts to be used in other rules or rule functions.

See *TIBCO BusinessEvents Event Stream Processing Query Developer's Guide.*

Pattern Language and Framework

The pattern language and framework are available with TIBCO BusinessEvents Event Stream Processing add-on software.

This add-on provides pattern-matching functionality, complementing TIBCO BusinessEvents rule processing and query processing features. Pattern Matcher consists of an easy-to-use language and a service that runs in a TIBCO BusinessEvents agent. It addresses some of the simpler and more commonly occurring problems in complex event processing such as patterns in event streams, correlation across event streams, temporal event sequence recognition, duplicate event suppression, and implementation of "Store and Forward" scenarios.

See *TIBCO BusinessEvents Event Stream Processing Pattern Matcher Developer's Guide*.

Deploy-time and Runtime Overview

A TIBCO BusinessEvents design-time project is deployed as a TIBCO BusinessEvents application.

When Cache object management (known as Cache OM) is used, the deployment can span multiple host servers.

You can use any of these deployment methods. It is recommended that you use only one method for each cluster you are deploying:

- At the command-line. You specify the CDD file to use and the processing unit within that CDD file.
- Using TIBCO Administrator. If you have been using this utility in your environment, you can continue to do so. (See in Hot Deployment in *TIBCO BusinessEvents Administration*.)
- Using TIBCO BusinessEvents Monitoring and Management. This is the preferred method.

All of the deployment methods use two resources: an EAR file and a cluster deployment descriptor, which is an XML file. To deploy using TIBCO BusinessEvents Monitoring and Management, you also need a site topology file.

The Enterprise Archive Resource (EAR) is the deployable version of a TIBCO BusinessEvents application. The EAR file contains runtime version of the project ontology, the channel definitions, the state machines (if TIBCO BusinessEvents Data Modeling add-on software is used), and so on. When you are finished designing the project in TIBCO BusinessEvents Studio, you simply choose a menu option to build the EAR.

Cluster Deployment Descriptor (CDD)

All methods of deployment require a cluster deployment descriptor (CDD).

Object Management

In the CDD, you configure the object manager you have chosen for the deployment.

Processing Units

Also in the CDD you configure the agents and *processing units* (engines) that will use the rules and ontology types you designed in your project.

Each engine equates to one processing unit, which runs in one JVM (Java Virtual Machine). One processing unit can host multiple agents, except in the case of a cache agent. A processing unit that hosts a cache agent cannot host any other agents. Each TIBCO BusinessEvents agent is a runtime component of the overall application.

Agent Configuration

Different kinds of agents play different roles in a large application. For example, inference agents perform rule evaluation, and cache agents manage the object instances generated and used by the inference agents (when the Cache object management type is used). To include multiple agents in an engine instance you add multiple TIBCO BusinessEvents agent classes in one processing unit.

Configuring an agent involves the following (depending on the type of agent you are configuring):

*   Selecting one or more sets of rules

*   Selecting destinations

*   Selecting event preprocessors for destinations, and thread settings to handle preprocessing more efficiently

*   Selecting functions that perform startup and shutdown actions

For more information about configuring the CDD see *TIBCO BusinessEvents Configuration Guide*.

Site Topology File

Before you can use MM for deployment, you define the site topology using the site topology editor in TIBCO BusinessEvents Studio.

You can also edit the underlying XML file directly. However, in TIBCO BusinessEvents Studio, the site topology editor introspects the CDD and EAR files to make useful project information available during configuration.

In the site topology editor you configure deployment units and host machines. Each deployment unit consists of one or more processing units, and related deploy-time settings. Each host machine is configured with details such as the software used for remote deployment. Then you bind deployment units to the desired host machines.

Deployment with TIBCO BusinessEvents Monitoring and Management (MM)

After you have configured MM to communicate with the cluster to be monitored, and you have defined the site topology file, you can log in and use the MM Console to deploy, stop and start, pause, and resume processing units.

Command-line tools for MM are available to perform basic deployment functions.

Monitoring and Management with MM

The MM component maintains a history of statistics, continuously queries the deployed engines for their status, and invokes methods on engines, as specified by the administrator.

Various overview panels and panes provide graphical views and alerts about the health of the cluster. You can configure the health thresholds as desired.

You can also use TIBCO Administrator features for monitoring and management. However they are not specialized for TIBCO BusinessEvents, whereas the MM component is specifically designed for use with TIBCO BusinessEvents.

TIBCO Hawk Application Management Interface

TIBCO BusinessEvents includes a set of TIBCO Hawk microagent methods that allow you to manage your TIBCO BusinessEvents deployment using TIBCO Hawk.

These functions are listed and described in TIBCO Hawk Microagent Methods in *TIBCO BusinessEvents Administration*.

TIBCO BusinessEvents Monitoring and Management provides a similar set of methods.

Hot Deployment

Depending on the changes made to your TIBCO BusinessEvents project, you may be able to replace an EAR file for a TIBCO BusinessEvents project with an updated one, without stopping the TIBCO BusinessEvents engine.

This feature is referred to as hot deployment. For more information about the TIBCO BusinessEvents hot deployment feature, including the project changes that are supported, see *TIBCO BusinessEvents Administration*.

Channels Destinations and Events

Channels (except for local channels) represent physical connections to a resource, such as a Rendezvous daemon, JMS server, HTTP server or client, Hawk domain, StreamBase server, FTL realm server, or a metaspace in TIBCO ActiveSpaces.

Destinations in a channel represent listeners to messages from that resource, and they can also send messages to the resource. All destinations for a particular channel use the same server. One project can have multiple channels of different types with multiple destinations as needed.

Arriving messages are transformed into simple events, using message data and metadata. Simple events sent out of TIBCO BusinessEvents are transformed to the appropriate type of message.

In addition to simple events, which work with incoming and outgoing messages of various sorts, TIBCO BusinessEvents uses a special-purpose event type called SOAPEvent, which inherits from SimpleEvent. It is used for sending and receiving SOAP messages in an HTTP channel.

Two other types of events are also used: time events and advisory events.

Channel Types

One project can have multiple channels of different types with multiple destinations.

You can either create an custom channel using the provided custom channel API or choose from the following types of channels:

ActiveSpaces 3.x channel

TIBCO ActiveSpaces software is a distributed in-memory data grid product. It can notify applications like TIBCO BusinessEvents of changes to the rows stored in a table which can be transformed into TIBCO BusinessEvents events. For more information, see the TIBCO ActiveSpaces documentation.

ActiveSpaces channel

They connect TIBCO BusinessEvents to TIBCO ActiveSpaces metaspace to enable it to monitor the activities in the metaspace, receive events from TIBCO ActiveSpaces and convert them into events in TIBCO BusinessEvents. A set of catalog functions are provided to control the type of the TIBCO ActiveSpaces events.

FTL channel

TIBCO FTL® messaging product is used for sending messages from one point to another. TIBCO BusinessEvents can send and receive events using TIBCO FTL as the medium. Using the FTL channel, TIBCO BusinessEvents can receive incoming TIBCO FTL messages and send TIBCO FTL messages across TIBCO BusinessEvents. You can use catalog functions in BusinessEvents to communicate with FTL.

Hawk channel

They connect TIBCO BusinessEvents to TIBCO Hawk domain to enable it to receive events from the Hawk monitor and transform them into events. A set of catalog functions are also provided which are used to control the Hawk microagents through Hawk APIs.

HTTP channel, including SOAP support

An HTTP channel acts as an HTTP server at runtime. This enables TIBCO BusinessEvents to serve requests from clients, and to act as a client of other servers. Support for SOAP protocol is provided by these features (using SOAP over HTTP):

- A specialized event type with a payload is configured as a skeleton SOAP message.

- A set of functions for extracting information from SOAP request messages and constructing response messages.

- A utility that constructs project resources based on the SOAP service's WSDL file (document style WSDLs with literal encoding are supported).

JMS channel

They connect TIBCO BusinessEvents to TIBCO Enterprise Message Service provider sources and sinks.

Each JMS Input Destination Runs a Session. Every JMS destination that is configured to be an input destination runs in its own JMS Session. This provides good throughput on queues and topics for processing, and less connections.

Kafka channel

To send and receive messages from a Kafka broker TIBCO BusinessEvents uses Kafka channel. The Kafka channel converts the incoming Kafka messages to the BusinessEvents events and output BusinessEvents events to outgoing Kafka messages.

Kinesis channel

Amazon Kinesis Data Streams helps in real-time collection and processing of data records. By using the Amazon Kinesis channel, TIBCO BusinessEvents can convert Kinesis data streams to TIBCO BusinessEvents events.

Local channel

They connect co-located agents at runtime.

MQTT channel

MQTT is a machine-to-machine connectivity protocol that enables remote connections for IoT applications. By using the MQTT channel, TIBCO BusinessEvents can receive MQTT messages and transform them into TIBCO BusinessEvents events.

StreamBase channel

Using the StreamBase Channel a BusinessEvents application (StreamBase client) can connect to a StreamBase server. The BusinessEvents application can subscribe to StreamBase streams to receive (dequeue) messages from the StreamBase server and send (enqueue) messages to the StreamBase server.

TCP channel

They connect to data sources not otherwise available through channels, using a catalog of functions.

TIBCO Rendezvous channel

They connect TIBCO BusinessEvents to TIBCO Rendezvous sources and sinks.

See *TIBCO BusinessEvents Developer's Guide* for more information on how to create a custom channel for your project.

Channel Serializers

For each type of channel (except local channels), TIBCO BusinessEvents uses a serializer to convert events to messages and a deserializer to convert incoming messages to events.

Local channels do not require serializers. HTTP channels also provides you the option of using action rule functions on the message instead of converting messages to event using deserializer.

Serializer and Deserializer Behavior
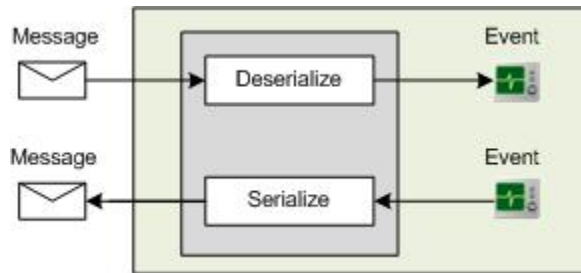
When you configure a destination, you select the appropriate serializer. (It actually includes the serializer and deserializer).

Deserializing from Rendezvous Message to Event

First level Rendezvous property values are used as values for matching event properties. Any additional (non-matching) Rendezvous properties are ignored.

The _payload_ field contents are passed into the event payload. Supported _payload_ field datatypes and Rendezvous wire format types are as follows:

Data Type	Wire Format Type
String	TibrvMsg.STRING
TibrvXML	TibrvMsg.XML
byte[]	TibrvMsg.OPAQUE TibrvMsg.I8ARRAY

If the event defines a payload, but the incoming Rendezvous message does not have a _payload_ field, TIBCO BusinessEvents attempts to map the entire message as the event payload.

Serializing from Event to Rendezvous Message

Event properties are transformed to first level Rendezvous message properties.

The event payload is passed to the Rendezvous message _payload_ field.

If the _payload_ field is of an unsupported type, or is missing, or if the event has not been configured for a payload, the payload is ignored.

First level Rendezvous property values are used as values for matching event properties. Any additional (non-matching) Rendezvous properties are ignored.

Message Acknowledgement

For each message type (that is, each type of channel), TIBCO BusinessEvents acknowledges the receipt of messages according to the protocol of the messaging system.

Some messages do not require acknowledgement. For example reliable Rendezvous messages do not require acknowledgement.

JMS messages might require acknowledgement, depending on the message acknowledgement mode (see JMS Message Acknowledgement Mode in *TIBCO BusinessEvents Developer's Guide* for a list of modes).

An event is acknowledged as follows:

• In a preprocessor: Immediately after the event is consumed.

• During a run to completion (RTC) cycle:

- With Cache OM, after the post RTC phase.
- With In Memory OM, after the post RTC phase, but only if the event has been explicitly consumed.

Events

Arriving messages are transformed into simple events, using message data and metadata.

Simple events sent out of TIBCO BusinessEvents are transformed to the appropriate type of message.

In addition to simple events, which work with incoming and outgoing messages of various sorts, TIBCO BusinessEvents uses a special-purpose event type called SOAPEvent, which inherits from SimpleEvent. It is used for sending and receiving SOAP messages in an HTTP channel. Two other types of events are also used: time events and advisory events.

TIBCO BusinessEvents processes three kinds of events:

Simple Event

A representation of a single activity (usually a business activity) that occurred at a single point in time. The SOAPEvent event type inherits from SimpleEvent.

Time Event

A timer. Time events can be configured to repeat at intervals, or they can be scheduled using a function in a rule or rule function.

Advisory Event

A notice generated by TIBCO BusinessEvents to report an activity in the engine, for example, an exception.

TIBCO BusinessEvents creates instances of simple events and time events based on user-configured event definitions.

Inheritance

You can use inheritance when defining simple events.

Attributes

In addition to user-defined properties, events have built-in attributes for use in rules and rule functions. For example, simple events have these attributes: @id, @extId, @ttl, and @payload. Concepts and scorecards also have built-in attributes. See *TIBCO BusinessEvents Developer's Guide* for details.

Simple Events

A *simple event definition* is a set of properties related to a given activity, while a *simple event* is an instance of a simple event definition.

Simple Event Definition

The simple event definition includes information for evaluation by rules, meta-data that provides context, and a separate payload — a set of data relevant to the activity. For example, suppose you are interested in monitoring the creation of new employee records. You might create a simple event definition that includes important fields from the employee record, perhaps the social security number, department, and salary. You could then write a rule to create an instance of this simple event each time a new employee record is created.

Simple Event

The simple event is an instance of a simple event definition and a record of a single activity that occurred at a single point in time.

Just as you cannot change the fact that a given activity occurred, once an event is asserted into the Rete network, you can no longer change it. (Before assertion you can use an event preprocessor to enrich the

event, however.) Simple events expire when their time to live has elapsed, unless TIBCO BusinessEvents has instructions to consume them prior to that time.

- Example 1: A temperature sensor records a reading that is above a predefined limit. The payload might include the temperature-sensor ID, the reading, and the date and time. This simple event might trigger a complex event that would immediately notify a manager.

- Example 2: A customer purchases four airline tickets from San Francisco, California to San Jose, Costa Rica. The payload might include the date and time of purchase, the date and time of the flight, the purchase price, the credit card number, the flight number, the names of the four passengers, and the seat assignments. This simple event alone may include no exceptions. However, it is possible that when examined within the context of other related events, an exception may arise. For example, one or more of the passengers may have booked tickets on another flight during the same time period.

Time Events

A time event is an event definition that triggers the creation of event instances based on time.

There are two ways to configure a time event:

Rule based

A rule schedules the creation of a time-event instance at a given time.

Time-interval based (Repeat Every)

TIBCO BusinessEvents creates a time-event instance at regular intervals.

> Time events do not go through an event preprocessor. If you are using cache-only cache mode, ensure that any objects are properly loaded. Events scheduled using scheduler functions, however, are sent through channels and would therefore go through event preprocessors in the usual way.

Scheduled Time Events

TIBCO BusinessEvents offers two kinds of time events, repeating and rule-based. In addition you can schedule events using functions.

> Time events configured to repeat at intervals are not supported in multiple-agent (multi-engine) configurations. Rule-based time events, however, are supported.

You can configure a time event to repeat at a configurable time interval. For example, if you configure a time event to repeat every thirty seconds, then every thirty seconds TIBCO BusinessEvents creates a new time event of that type.

You can configure a repeating time event to create a specified number of events at each interval. The time interval begins during engine startup. See Engine Startup and Shutdown Sequence in *TIBCO BusinessEvents Administration* for specific details.

Rule-Based Time Events

A rule-based TimeEvent resource has only a name and description.

You can then use it in a rule to schedule a simple event to be asserted, using its ontology function, Schedule*TimeEventName*() in a rule. You can schedule the event to be asserted after a period of time, and you can pass information to the event and specify its time to live. You can call the Schedule*TimeEventName*() function in different places with different time delays.

You can use rule-based time events in various ways. For example, you might write rules that check for delays in order fulfillment:

1. A new Order event is asserted, and Rule A (which has Order in its scope) creates a time event T and configures it to be asserted in sixty minutes, and passes the order ID as the closure parameter value. (Rule A also sends the order details to another system.)

2. Sixty minutes after Rule A executes, timer event T is asserted.

3. The assertion of time event T triggers Rule B, which has T in its scope. Rule B checks the order status. If the order is delayed, it sends out an alert.

Advisory Events

Advisory events are asserted into the Rete network automatically when certain conditions, for example, exceptions, occur.

Add the AdvisoryEvent event type to rules to be notified of such conditions. An advisory event expires after the completion of the first RTC cycle (that is, the time to live code is set internally to zero).

The TIBCO BusinessEvents engine automatically asserts an advisory event when it catches an exception that originates in user code but that is not caught with the catch command of the TIBCO BusinessEvents Exception type. For information on working with other kinds of exceptions, see the "Exception Handling" section in the *TIBCO BusinessEvents Developer's Guide*.

Advisory events are also used in the container mode TIBCO BusinessEvents-ActiveMatrix BusinessWorks integration feature invokeProcess() function. Such events are asserted when the ActiveMatrix BusinessWorks process fails or times out (or is cancelled).

An advisory event (engine.primary.activated) is asserted when an engine has finished starting up and executing startup functions, if any (see Engine Startup and Shutdown Sequence in *TIBCO BusinessEvents Administration*).

Default Destinations and Default Events

Using default destinations and default events simplifies project configuration for many scenarios.

Incoming messages can be mapped to a default event that you specify when you configure the destination. All messages arriving at that destination are transformed into the default event type, unless they specify a different event.

A channel is configured to listen to the flow of messages on Rendezvous. The *orders* destination directs TIBCO BusinessEvents to map messages coming in on the subject, *orders*, to the *new_order* simple event.

You can map incoming messages to specified event types. The technique is explained in Working with Rendezvous Channels in *TIBCO BusinessEvents Developer's Guide*.

Outgoing messages can be sent to a default destination. When the destination is not otherwise specified (for example in rules or rule functions), events are sent to the destination you select as their default destination.

The event credit_timeout is sent out through its default destination credit.

You can send an event to the default destination of its event type using the Event.sendEvent() or Event.replyEvent() functions.

You can send an event to a specified destination using the Event.RouteTo() function.

Mapping Incoming Messages to Non-default Events

Incoming messages can be mapped to default events or to specified event types.

The fields in a message header instruct TIBCO BusinessEvents to map the incoming message to a specified event type:

- The field named _ns_ takes a namespace as a value. The namespace points to the event type, for example, www.tibco.com/be/ontology/Events/MyEvent

- The field named _nm_ takes the name of the event, for example, NewMyEvent

- The field named _extid_ takes the unique external id of the event.

These fields are added and filled automatically for events created using TIBCO BusinessEvents rules. You can also add these fields to the incoming messages from other sources if you have control of those

messages. You can also use the **Include Event Type** field in the destination of channel to suppress the original behavior of including _ns_ and _nm_ fields during serialization and deserialization.

Time to Live and Expiry Actions

Events have a time to live (TTL) setting. Events cannot be modified after they are initially asserted, but they can continue to trigger rules during their time to live.

Global Variables are supported for Simple Event, TTL and Timer Event, Repeat every Values.

When Cache object management is used, events with a sufficiently long time to live (TTL) setting are cached.

With Cache OM types, the TTL period is re-evaluated when an event is reloaded from cache. For example, if the TTL is 60 minutes and the event is reloaded 30 minutes after it is asserted, then its remaining TTL is 30 minutes.

Set the event's time to live so that it can trigger the rules you intend. If a rule correlates different events, you must ensure that those event instances are all in the Rete network concurrently. Time to live options are as follows:

- Zero (0): the event expires after the completion of the first RTC cycle. Do not set to 0 if you want to correlate the event with a future event or other future occurrences of this event, as explained below.

- One or higher (>0): the event expires after the specified time period has elapsed. The TTL timer starts at the end of the action block of the rule or preprocessor function in which the event is first asserted.

- A negative integer (<0): the event does not expire, and must be explicitly consumed.

Cache OM and Event Deletion with Cache OM, events are locally consumed when Event.consumeEvent() is called, but the event is not removed from cache until the post RTC phase.

Example

Consider the following example:

- A process sends eventA, eventB, and eventC.

- The TTL for all three simple events is 0.

- Rule 1 has the condition: `eventA.id == eventB.id`.

- Rule 2 has the condition: `eventC.id != null`.

At runtime, TIBCO BusinessEvents behaves as follows:

- TIBCO BusinessEvents receives `eventA`. Because there is no `eventB` in the Rete network, `eventA` doesn't trigger any rules. TIBCO BusinessEvents consumes `eventA`.

- TIBCO BusinessEvents receives `eventB`, but `eventA` has been consumed — there is no `eventA` in the Rete network. So `eventB` does not trigger any rules. TIBCO BusinessEvents consumes `eventB`.

- TIBCO BusinessEvents receives `eventC`, which triggers Rule 2 because Rule 2 depends only on `eventC`.

To trigger Rule 1, you must configure the time to live for `eventA` and `eventB` to ensure that both events will be in the Rete network concurrently. You can trigger Rule 1 in these ways:

- If you know that `eventA` is sent before eventB, set the TTL for `eventA` to a time greater than the maximum period that can elapse between sending eventA and sending `eventB`.

- If you do not know the order in which `eventA` and `eventB` are sent, set the TTL for both simple events to a time greater than the maximum time between the occurrence of the two simple events.

Event Expiration and Expiry Actions

After the time to live (TTL) period, the event expires and is deleted from the Rete network. Any expiry actions are taken.

With Cache object management, events TTL is evaluated when the event is retrieved from the cache.

For each simple event definition, TIBCO BusinessEvents allows you to specify one or more actions to take when the event expires, using the TIBCO BusinessEvents rule language. For example, you can write an action that routes the simple event to a different destination, sends a message, or creates a new event. This action can be anything that is possible with the rule language.

An expiry action can be inherited from the event's parent.

> If an event is explicitly consumed in a rule, TIBCO BusinessEvents does not execute the expiry action.

Event Preprocessors

Event preprocessors are rule functions with one argument of type simple event.

Event preprocessors are not used for time or advisory events, and they are multithreaded.

An event preprocessor is assigned to a destination and acts on all events arriving at that destination. Event preprocessors perform tasks after an incoming message arrives at the destination and is transformed into a simple event, but before it is asserted into the Rete network (if it is — events can be consumed in the event preprocessor).

> - If you are using Cache Only mode, take care when designing rules that execute as a result of a time event. For example, a rule that has a join condition using a time event and a concept would not execute if the concept is not loaded in the Rete network and so should not be used in an event preprocessor if the concept is configured as Cache Only mode.
> - Events consumed in a preprocessor are acknowledged immediately (and not after the post RTC phase).

You can aggregate events, edit events, and perform other kinds of event enrichment in a preprocessor. You can also use preprocessors as explained below.

You must set locks in the preprocessor when concurrency features are used to protect concept instances during RTC. Locking ensures that updates to concept instances during an RTC do not conflict with another set of updates to the same concept instances in another RTC. Locks are released at the end of the RTC.

If you are using the Cache Only mode for any entities, you must also load the relevant entities from the cache using an event preprocessor.

You can also use preprocessors to improve performance by avoiding unnecessary RTCs in the inference engine. For example you can consume events that are not needed. Another way to use the preprocessor for efficient processing is to transfer an event's contents to a new concept that is not processed by the agent's set of locally active rules. Such a concept is automatically asserted, and does not trigger rules. It is saved into the cache (depending on OM configuration) where it is available for processing by any agent as needed.

Preprocessor Usage Guidelines

Keep in mind several guidelines for using preprocessors.

Consuming events in a preprocessor is allowed

It can be useful in some applications and reduces the flow of messages into the Rete network. Such events are acknowledged immediately (if they require acknowledgement).

You can only modify events before they are asserted into the Rete network

Rule evaluation depends on event values at time of assertion, so values can be changed only before assertion, that is, in the preprocessor.

You can create concepts but not modify existing concepts

Modifying concepts that already exist in the system could disrupt an RTC. You can modify concepts that were created in the same preprocessor, however. You cannot add a existing concept as a child to a newly created concept in preprocessor, as it modifies the existing concept.

Concepts created in a preprocessor are not asserted until the RTC starts. So, for example, after one event preprocessor ends and before its RTC begins, no other preprocessor can access the new concept.

Concepts

Concept types are descriptive entities similar to the object-oriented concept of a class.

Concept types describe a set of properties, such as one concept might be Department and include department name, manager, and employee properties.

You can add concept definitions so that information that arrives in events or from other sources can be organized and persisted as needed, and used in rules. You can add definitions manually. You can also import database tables as concept definitions. Concept instances are created in rules.

Rules at runtime can create instances of concepts. For example, when a simple event arrives, a rule can create an instance of a concept using values present in the event. Rules can also modify existing concept instance property values.

Concepts must be explicitly deleted from working memory when no longer needed or they will steadily increase memory usage. Use the function `Instance.deleteInstance()` to delete concept instances.

Depending on other factors, adding, modifying, and deleting concept instances can cause TIBCO BusinessEvents to evaluate or re-evaluate dependent rules, as explained in Conflict Resolution and Run to Completion Cycles.

Concepts are automatically asserted into the Rete network when created, except in the following cases:

*   Database concepts returned by database query operations (requires TIBCO BusinessEvents Data Modeling).
*   Concepts passed to a rule function in the context of ActiveMatrix BusinessWorks integration projects.

Each concept property includes a history, the size of which is configurable. The history size determines how many previous values TIBCO BusinessEvents stores for that property. See Concept Property History.

Database concept properties do not support history tracking (which is available in TIBCO BusinessEvents Data Modeling).

Concept Relationships

Concepts can have inheritance, containment and reference relationships with other concepts. See Inheritance Relationships.

Exporting Concepts to XSD Files

You can export concept and event types to XML Schema Definition (XSD) files. XML schemas are used for interoperability between TIBCO BusinessEvents and third-party tools or SOA platforms that use well-defined XML for message communication, transformation, and validation.

Concept Serialization and Handling of Null Value Properties at Runtime

By default, when concept instance objects are serialized to XML, properties with null values are excluded. You can change this behavior so that null values are included. You can also change the XSD for a concept object to allow null values, using the nullable attribute.

Concepts and State Machines

If you are using the TIBCO BusinessEvents Data Modeling add-on product, you can associate a concept with a state machine. See *TIBCO BusinessEvents Data Modeling Developer's Guide* for details.

Concept Relationships

You can learn more information about concept relationships, such as inheritance relationships and reference relationships.

Concept Property History

Each concept property includes a history, the size of which is configurable.

The history size determines how many previous values TIBCO BusinessEvents stores for that property. You can also set the history policy to record all values or only changed values.

- ConceptReference Properties History is tracked when a contained or referenced concept instance changes to a different concept instance. History is *not* tracked, however, when a contained or referenced concept's properties change. See Inheritance Relationships for more on containment and reference relationships.
- Database concept properties do not support history tracking.

If you set the history size to one or more, TIBCO BusinessEvents stores the property value when the property changes, along with a date and timestamp, up to the number specified. When the maximum history size is reached, the oldest values are discarded as new values are recorded.

If you set the history size to 0, TIBCO BusinessEvents does not store historical values for the concept. It stores the value without a time and date stamp.

For example, consider a Customer concept:

Customer Concept

Property Name	History	Comments
customer_name	1	These properties tend to be very stable and you may have little interest in tracking a history for them.
customer_address	1	
city	1	
state	1	
zip	1	
account_number	0	With history size 0, TIBCO BusinessEvents does not record the timestamp when the value is set.
credit_limit	4	Credit limit may change more frequently and you may have an interest in tracking the changes.

Historical Values are Stored in a Ring Buffer

The historical values for a concept property are kept in a ring buffer. The ring buffer stores both the value and the time at which the value was recorded. After the ring buffer reaches maximum capacity, which is eight in this example, TIBCO BusinessEvents begins replacing older values such that it always stores the n most recent values, where n is the history size.

TIBCO BusinessEvents can record values using either of these policies:

- **Changes Only** - TIBCO BusinessEvents records the value of the property every time it changes to a new value.

- **All Values** - TIBCO BusinessEvents records the value of the property every time an action sets the value even if the new value is the same as the old value.

Which you choose depends on what you are tracking. For example, if you are setting the history for a property that tracks how many people pass a sensor every five minutes, All Values might be the best policy. However, if you are setting the history for a property that tracks the level of liquid in a coffee pot, Changes Only might be more appropriate.

The history policy affects how frequently TIBCO BusinessEvents re-evaluates rules that are dependent on the property. Each time TIBCO BusinessEvents records a value, it reevaluates rules that are dependent on that property. If you track changes only, rules are re-evaluated less frequently than if you track all values.

Containment Relationships

Containment relationships allow one concept to be contained inside another concept.

You can define a hierarchy of containment, where each concept in the hierarchy is contained by the concept above it. The relationship is defined using a `ContainedConcept` property in the container concept.

When working with container and contained concepts in the rule editor, the XSLT mapper and XPath builder show the entire hierarchy of properties. In the rule editor, you can also use the `@parent` attribute to get the parent of a contained concept instance.

Deep containment relationships can cause memory issues. When TIBCO BusinessEvents retrieves a concept from cache, its child concepts are also retrieved. When you modify a child concept, its parent concepts are considered to be modified. It is recommended that you keep concept relationships shallow.

Containment and Reference Concept Relationship Rules

Containment and Reference Concept Relationship Rules

Containment One concept is contained in another	Reference One concept points to another
Designtime Rules	
One container concept can contain multiple different contained concepts, and a contained concept can itself also act as a container concept. One referring concept (that is, the concept that has the `ConceptReference` property) can have a reference relationship with multiple referenced concepts, and a referenced concept can also refer to other concepts.	
A container concept can link to a contained concept using only one `ContainedConcept property`. (Some other object-oriented languages do allow you to reuse object types in parent object properties.)	A referring concept links to a referenced concept using multiple `ConceptReference` properties. (That is, multiple `ConceptReference` properties can reference the same referenced concept.)
A contained concept can have only one container concept.	A referenced concept can be referred to by multiple referring concepts
Runtime Rules	

Containment **One concept is contained in another**	Reference **One concept points to another**
When one contained instance is replaced with another: TIBCO BusinessEvents automatically deletes the instance that it replaced. You do not have to delete the replaced instance explicitly.	**When one referenced instance is replaced with another:** TIBCO BusinessEvents does *not* delete the instance that it replaced automatically. It may not be appropriate to delete the referenced instance. If you want to delete the referenced instance, do so explicitly.
When a contained instance is modified: The container instance is also considered to be modified. The reasoning can be seen by a simple example: a change to the wheel of a car is also a change to the car. Rules that test for modified instances would return the `Car` concept instance as well as the `Wheel` concept instance.	**When a referenced instance is modified:** The referring instance is *not* considered to be modified. The reasoning can be seen by a simple example: a change to the support contract for a customer is not a change to an order that references that customer.
When a container instance is asserted or deleted: The contained instance is also asserted or deleted, along with any other contained instances at lower levels of the containment hierarchy.	**When a referring instance is asserted or deleted:** The referenced instance is *not* also asserted or deleted.

See the provided Containment Example.

Containment Example

This example shows how to configure a concept `Car` to contain a concept `Wheel` by adding a `ContainedConcept` property `Wheels`, whose value is an instance of the concept `Wheel`.

The `Wheels` property provides the link between the container and contained concept:

Car (Concept) — `Wheels` (property) — Wheel (Concept)

The concept `Car` contains four instances of the contained concept `Wheel`, so you define the property as an array. The concept `Car` could also contain other concepts, such as `Door` and `Engine`, defined in a similar way using `ContainedConcept` properties.

However, the contained concepts — `Wheel`, `Door`, and `Engine` — cannot be contained by any other concept type. They can only be contained by the `Car` concept. For example, the concept `Wheel` cannot be contained in the concept `Motorbike`, if it is already contained by the concept `Car`.

A container concept can link to a contained concept using only *one* `ContainedConcept property`. You can use inheritance, however, to achieve a result similar to that gained by the general programming technique of linking to multiple contained class properties. Suppose you extend the concept `Wheel` by creating child concepts `CarWheel` and `MotorcycleWheel`. You can then use `CarWheel` as the concept contained by `Car`, and `MotorcycleWheel` as the concept contained by `Motorcycle`. Rules that apply to `Wheel` also apply to `CarWheel` and `MotorcycleWheel`, because of inheritance.

Depending on your needs, another option would be to use a reference relationship instead of a containment or inheritance relationship.

Inheritance Relationships

Inheritance relationship provide that a concept inherits all the properties of another concept, similar to Java, where a subclass inherits all the properties of the superclass that it extends.

You can define a hierarchy of inheritance, where each concept in the hierarchy extends the concept above it. The relationship is defined by the Inherits From field in the concept resource editor. In DataGrid, the child concepts also inherits the indexes defined in the parent concepts.

Concepts that are related to each other directly or indirectly by inheritance cannot have distinct properties that share a common name. Therefore, the following restrictions apply:

- If two concepts are related by inheritance, you cannot create a new property in one with a name that already exists in the other.
- If two unrelated concepts have properties that share a name, you cannot create an inheritance relationship between the two concepts.

TIBCO BusinessEvents does not allow you to create an inheritance loop; for example, if Concept A inherits from Concept B, Concept B cannot inherit from Concept A. At runtime, a rule on a parent concept also affects all its child concepts. For example, suppose the concept Coupe inherits from the concept Car. A rule on Car is therefore also a rule on Coupe.

Reference Relationships

In a reference relationship, one concept instance references another concept instance.

A concept that is the target of a reference can itself refer to one or more other concepts. Reference relationships are not, however, hierarchical.

The relationship is defined by a `ConceptReference` property in the referring concept.

See Containment and Reference Concept Relationship Rules for rules governing the behavior of concepts linked by containment or reference. The table also helps you to choose which is the appropriate type of relationship for your needs.

Properties of concept references cannot be used in a condition.

Reference Example: Order with SalesRep and Customer

To configure a concept `Order` to reference a concept `SalesRep`, you add a `ConceptReference` property, `Rep` for example, whose value is the ID of concept `SalesRep`. The `Rep` property provides the link between the referring and referenced concepts:

- `Order` (Concept) — `Rep` (property) — `SalesRep` (Concept)

You can also define additional reference relationships such as:

- `Order` (Concept) — `BackupRep` (property) — `SalesRep` (Concept)
- `Order` (Concept) — `Lines` (property array) — `LineItem` (Concept)
- `Order` (Concept) — `Cust` (property) — `Customer` (Concept)
- `Customer` (Concept) — `Orders` (property) — `SalesRep` (Concept)

Reference Examples: Self Reference

A concept definition can have a reference relationship to itself. This is generally because the instances of one concept definition can have reference relationships to other instances of the same definition. For example:

- A `ListItem` concept has a `next` property which is a reference to a `ListItem` concept.

- A `Person` concept has a `spouse` property which is a reference to a `Person` concept.
- A `Person` concept has a `children` property which is an array of references to Person concepts.

When a Contained or Referred Concept Instance is Deleted

There is an important difference in behavior when history is tracked for an array property, and when history is not tracked, or the property is not an array.

Effect of deleting a contained or referenced concept:

ContainedConcept or ConceptReference Property	Effect of deleting a Contained or Referenced Concept:
Single value property, regardless of history setting.	The value of the `ContainedConcept` or `ConceptReference` property becomes null.
Multiple-value property (array), with History is set to 0 or 1 (historical values are not tracked).	The array entry that held the deleted concept is removed, reducing the array size by one. Delete higher position numbers before lower position numbers to ensure the correct entries are deleted. The array entry that held the deleted concept is removed, reducing the array size by one, and reducing by one the index of every entry in the array at a higher index than the deleted one. (When deleting multiple entries at once, delete higher position numbers before lower position numbers to ensure the correct entries are deleted.)
Multiple-value property (array), whose History is set to 2 or more (historical values are tracked).	The array entry that held the deleted concept remains and its value is set to null, so that history can be tracked.

Scorecards

A scorecard is a special type of concept that serves as a set of static variables available throughout the project.

You can use a scorecard resource to track key performance indicators or any other information.

Unlike concepts and events, each scorecard resource is itself a single instance — it is not a description for creation of instances. You create the scorecard at design time. Its values can be viewed and updated using rules.

It is more accurate to say there is one instance of a scorecard per inference agent. Each inference agent in an application has its own instance of the score card. Scorecards are not shared between agents.

Any agent that uses scorecards, and also uses Cache OM, must be assigned a unique key so that the correct scorecard can be retrieved from the cache. The key is set in the Processing Unit tab of the CDD.

It is not necessary to add scorecards to the declaration of a rule. Because there is only one instance of each scorecard in a deployed TIBCO BusinessEvents agent, any change causes all rules that use the scorecard in their conditions to be evaluated.

The Instance.isModified() function works differently with scorecards than with concepts. There is only one instance of a scorecard per agent, rather than one per RTC. So after a scorecard is modified it will return true until the agent is restarted.

(In the case of a concept instance, Instance.isModified() returns true after the instance has been modified only for the rest of the RTC in which it is modified.)

Rules

Most rules in TIBCO BusinessEvents are used for inferencing. However, regular business rules also have a role to play.

A TIBCO BusinessEvents rule has three components:

Declaration

They declare which concepts and events the rule will depend on, and the names by which instances of these entities can be referred to in the conditions and actions. Aliases must be valid identifiers. Declaring multiple terms of the same type allows the rule to consider multiple instances of the corresponding entity.

Conditions

Each statement in the condition must evaluate to a boolean value. All of these statements must be true for the rule's action to be executed. Assignments and calls to certain functions are disallowed in the condition.

Actions

List of statements that will be executed, when the rule is fired, for each combination of terms that matches all the conditions.

You can organize rules depending on your project and project maintenance needs. Rules are organized in folders. At deploy time you can select folders of rules or individual rules (or both) for deployment.

Inferencing rules are at the heart of TIBCO BusinessEvents. Inferencing rules are declarative, and at runtime are executed based on the outcome of each conflict resolution cycle. Statements in a rule action might create or modify concept instances, create and send simple events, call functions and rule functions, and so on depending on need.

Rule Priority and Rank

For each RTC, the rule agenda is sorted by priority and then within priority by rank, for those rules that use the same ranking mechanism.

Use of priority and rank is optional. You can also use priority without using rank.

TIBCO recommends that you use priority and rank features only as needed; that is, unless there is reason to set priority (or priority and rank), let the rule engine determine the sequence of execution. This lessens the complexity of rule maintenance, and takes advantage of the power of the inferencing engine.

Rule Priority

Because TIBCO BusinessEvents rules are declarative rather than procedural, there is no inherent order for processing. However, a priority property allows you to specify the order in which rules in one RTC execute.

Rule Rank Within the Same Priority

If you want to also control the order in which rules with the same *priority* execute, you can use the rule rank feature. The value for the Rank property is a rule function that returns a double. The larger the return value, the higher the ranking. You can specify the same rule function in different rules to perform ranking across tuples of those rules.

Other Rules

Not all rules in TIBCO BusinessEvents are inferencing rules. Rules in decision tables are business rules, executed only when the table is invoked.

Form-based and Source Rule Editors

When you work with rules and rule functions, you can choose how to work:

- Using a form-based rule editor, similar to the rule editor in earlier versions of TIBCO BusinessEvents

- Using a source editor, which is closer to a Java programming environment.

You can switch between editors and changes made in one editor are reflected in the other one. You cannot switch from the source editor to the form editor if there are validation errors in the code.

Effect of Cache Only Cache Mode

When using Cache Only cache mode for one or more entities, you must consider how to handle the cache-only entities when you write rules and preprocessor rule functions. See Working With Cache Modes and Loading Cache Only Objects into the Rete Network in *TIBCO BusinessEvents Architect's Guide*.

Rule Functions

A rule function is a function written in the TIBCO BusinessEvents rule language. All rule functions created for a project are available project-wide.

Rule functions can take arguments and can return a value. The return type can be set to void, indicating that the rule function does not return anything. Like other types of functions, you can use rule functions in rule conditions and rule actions.

You can use project settings to use rule functions as preprocessors (see Event Preprocessors and as startup and shutdown actions.

Virtual Rule Functions and Decision Tables

Decision tables are available with TIBCO BusinessEvents Decision Manager add-on software.

A Virtual Rule Function (VRF) has arguments but no body or return type. The implementation of a virtual rule function is a decision table. Business users can create decision tables in the TIBCO BusinessEvents Decision Manager stand-alone business user interface. Decision tables can also be created in the TIBCO BusinessEvents user interface.

Users start by selecting a VRF. They drag and drop entities from an argument explorer to form rows in a decision table. Each row forms a business rule, for example the condition area might specify that age is less than 18, and the action area might specify that credit is refused. More technical users can use the TIBCO BusinessEvents rule language to create more complex rules.

One VRF can have multiple implementations. You can set a priority that determines the order of execution for multiple implementations of a VRF. Functions are also available for choosing an implementation to execute (and other actions specific to decision tables). If there is just one implementation, you can call the virtual rule function in the same way you call any other rule function.

Startup and Shutdown Rule Functions

Startup and shutdown rule functions are rule functions that are configured to execute during normal system startup and shutdown, respectively.

Startup and shutdown rule functions take no arguments and their Validity setting must be Action (meaning they cannot be used in conditions or queries).

See Engine Startup and Shutdown Sequence in TIBCO BusinessEvents Administration. Understanding this sequence helps you understand what you can do in startup and shutdown actions.

Startup Rule Functions

Startup rule functions are optional and are used to initialize the system. For example they can provide initial values for scorecards. Startup rule functions can be used to perform more "expensive" operations so that the system is more efficient at runtime. For example, in a startup rule function you might load specified entities from the backing store to the cache.

Startup rule functions may trigger rule actions. However, note that TIBCO BusinessEvents executes all startup rule functions before it begins the first RTC cycle, which completes when all rules eligible to execute have executed and no more actions remain.

Shutdown Rule Functions

Shutdown rule functions are optional and are used to perform various actions during a normal shutdown, for example, they can send events to external systems.

When Startup Rule Functions Execute

Startup rule functions execute on startup of an active node.

In recovery situations, startup rule functions execute on failback to a failed node that has restarted. However, if recovery is from a situation that does not involve node failure, then startup actions do not execute. For example, the network connection goes down. The agent becomes inactive and fails over to another node. The connection is restored. The agent becomes active again, but does not restart. Startup functions do not execute on the node that became active again.

If you want to execute startup rule functions on only one node in a deployment, use programming logic to do so.

Creating Entities With a Startup Action in a Multi-Engine Project

Startup (and shutdown) rule functions execute in all active agents. When multi-engine (multi-agent) functionality is used, ensure that multiple agents do not attempt to create the same entity.

ActiveMatrix BusinessWorks Containers

In ActiveMatrix BusinessWorks integration projects, if ActiveMatrix BusinessWorks is running as the container, do not specify any startup actions that result in starting or invoking an ActiveMatrix BusinessWorks process.

Note that after the ActiveMatrix BusinessWorks engine is initialized, processes that invoke TIBCO BusinessEvents rule functions will fail if the TIBCO BusinessEvents engine has not finished starting up. For example, an ActiveMatrix BusinessWorks process that listens to a JMS queue may attempt to invoke a TIBCO BusinessEvents rule function before the TIBCO BusinessEvents engine has started up.

Runtime Inferencing Behavior

At runtime, one or more nodes (JVMs) running one or more TIBCO BusinessEvents *inference agents* process the incoming events using a Rete network as the inferencing engine, and a set of rules that are triggered by conditions in incoming events. One or more event stream processing query agents can query incoming events.

TIBCO BusinessEvents has two layers of functionality:

Rules Evaluation and Execution

It is based on the state and value of objects and incoming events. This functionality is achieved using one or more inference agents configured with the appropriate rules. Each inference agent executes rules using one or more Rete networks to optimize performance and provide rule inferencing capabilities.

Lifecycle Management of Objects and Events

This includes distribution, clustering, persistence and recoverability. Various options are available to achieve the functionality appropriate for business needs: in-memory only storage of objects, use of a cache, and addition of a backing store (database).

Queries and Pattern Matching

In addition, when TIBCO BusinessEvents Event Stream Processing software is used, a third layer is added: queries and pattern matching. A query agent enables visibility into the event stream and cache data. Pattern matching features enable actions to be taken on recognition of a pattern of events, or failure to complete a pattern of events.

Rule Evaluation and Execution

Information from enterprise applications and other sources flows into TIBCO BusinessEvents through channels as messages.

Messages represent the events that TIBCO BusinessEvents processes based on event definitions (event types). Events can be filtered (ignored), preprocessed into concepts or cached concepts, or asserted into the rule engine's working memory.

In an inference agent, all the rules whose conditions match information in the events (as well as concepts, if specified in the rule conditions) are assembled into a *rule agenda* and the first rule executes. If a rule successfully executes, its rule actions create and modify the objects in working memory. The rule agenda is derived from an internal runtime memory structure known as a *Rete network* (because it uses a derivative of the *Rete algorithm*).

TIBCO BusinessEvents rule engine is a forward-chaining inferencing engine. Every time the facts (concepts, score cards, and events) in its working memory change — due to rule actions or the arrival of new events — the inferencing engine updates the rule agenda. As a result, new rules are available to execute while others are now unavailable. The selection of which rule to execute first from a choice of several is called *conflict resolution*. The agenda process repeats until there is no more new information to process. This is known as RTC, or *run to completion*.

Conflict Resolution and Run to Completion Cycles

To design rules more effectively, you need to understand what triggers rules to execute, and why a rule may not execute.

A *run to completion*, or RTC, cycle generally begins when an external action causes changes to the Rete network. It ends when there are no more rule actions to execute as a result of that initial change (and any subsequent changes caused by rule actions). This is also known as *forward chaining*, or *inferencing*.

During one RTC changes can occur in the Rete network, but no new external actions can affect it.

One RTC is composed of one or more *conflict resolution cycles*. A conflict resolution cycle begins when TIBCO BusinessEvents builds (or refreshes) a *rule action agenda*, a list of all rules that are eligible to fire. The agenda is used to determine which rule action to execute next. The agenda is built based on the following information:

- The scope and conditions of the rules in the project.
- The current contents of the Rete network.

One conflict resolution cycle ends when a rule action is executed (or the agenda is empty). If the rule action changes the contents of the Rete network, another conflict resolution cycle begins.

The Rete network changes

The first of the conflict resolution cycles is initiated by change in the Rete network, caused by an external action such as a message arriving at a destination. All subsequent changes to the Rete network during one RTC occur because of rule actions.

TIBCO BusinessEvents builds the agenda

To build the rule action agenda, TIBCO BusinessEvents examines all rules that are *newly true* because of the change to Rete network and compares them with rule dependencies. The agenda's entries are ordered according to rule priority, rule rank, and other criteria.

TIBCO BusinessEvents executes the first rule on the agenda and removes it from the agenda

As a result, one of the following occurs:

- The rule action does not change the Rete network and TIBCO BusinessEvents executes the next rule entry in the agenda (if there is one).

  OR

- The rule action does change the Rete network and TIBCO BusinessEvents refreshes the rule action agenda (see next section).

Events created during an RTC are not sent to destinations until the entire RTC is complete. Similarly, objects are not written to cache until the entire RTC is complete.

Next conflict resolution cycle

TIBCO BusinessEvents refreshes the rule action agenda

If a rule action changes the contents of the Rete network, the agenda is refreshed, beginning a new conflict resolution cycle. When the agenda is refreshed, any of the following can occur:

- Rules that have become newly true are added to the agenda.
- Rules that have become false are dropped from the agenda.
- Rules that were newly true at the last conflict resolution cycle and are still true remain in the agenda. (In other words, rules are newly true for the duration of the run to completion cycle unless they become false.)

As a result, either the agenda is empty and the RTC ends, or the first rule in the refreshed agenda is executed, ending this conflict resolution cycle and beginning the next one.

An empty agenda marks the end of one RTC

An empty agenda ends the RTC

At some point, no more actions remain to be executed. The conflict resolution has run to completion. The RTC is over. Now begins the post RTC phase. At various points during this phase the following actions happen (depending on how the project has been configured):

- Events are sent to destinations.

- Cache OM: Changes are saved to the cache and written to the backing store.

- Cache OM, Cache Only mode: All Cache Only objects are removed from the Rete network.

- Profiler: profiler data is updated.

How to Work with Rules

Before any data enters the system, TIBCO BusinessEvents builds the Rete network, which embodies all the *rule dependencies*, using the rule conditions (if any).

All the dependencies in a rule are called its *dependency set*.

For example, a rule has this condition:

```
c.name == "Bob";
```

Where c is a concept of type /Customer. In this case, the dependency set of the rule contains only the name property of the concept type /Customer.

As another example, suppose a rule has these conditions:

```
b.num<10;
hasAGoldMembership(c);
```

where b is another concept type and num is one of its properties. The dependency set for this rule is b.num and c.

How to Test the Truth of a Rule's Conditions Using the Dependency Set

During a conflict resolution cycle, TIBCO BusinessEvents tests each rule's dependency set against the new set of facts. If the facts match the rule dependencies, the rule conditions are all true and the rule action is added to the rule action agenda. The structure of the Rete network enables very quick matching between facts and rule dependency sets.

If TIBCO BusinessEvents cannot calculate dependencies on the properties of an entity from the rule condition, for example if you pass an entity to a function, TIBCO BusinessEvents evaluates the rule every time the entity or its properties changes.

How a Rule Becomes Newly True

A rule is true if objects in the rule scope exist in the Rete network and if all of the rule conditions are met by objects in the Rete network. However when building the rule action agenda, TIBCO BusinessEvents examines only rules that are *newly true*.

A rule is *newly true* if it has become true due to a change in the Rete network during the current RTC.

In the case of a rule with no conditions, assertion of an object in the scope (declaration) of the rule makes the rule newly true.

A rule that was false and becomes true because of the changes in the Rete network during the RTC is newly true.

Less obviously, a rule that was already true can also become newly true. For example, a rule may already be true because a condition that specifies a range is satisfied. It becomes newly true if the property value in the Rete network changes but is still within the range. For example, the condition c.b<10; is true if the Rete network includes a c.b with value 9. It is newly true if an action at the end of a conflict resolution cycle changes the value from 9 to 8.

A rule remains newly true until it is executed or it is removed from the agenda, or the RTC ends.

A rule is removed from the agenda because a change in the Rete network during an RTC means that the facts no longer satisfy its dependency set, for example because a concept instance is deleted or a concept property changes value.

Order of Evaluation of Rule Conditions

The order in which conditions are evaluated is determined internally by TIBCO BusinessEvents.

Using a rule's dependency set, TIBCO BusinessEvents evaluates the following kinds of rule conditions in the order shown, to perform the evaluation efficiently:

A. Filters

These are conditions that only involve one scope element (object). Filters are the least expensive operations, in terms of processing cost. For example:

```
Customer.type == "gold";
Customer.numOrders > 50;
```

B. Equivalent join conditions

These are conditions that compare two expressions using == where each expression involves one (different) object from the scope. Equivalent joins take more processing than filters, but less than non-equivalent joins. For example:

```
Customer.accountMgr == AccountManager.id;
```

C. Non-equivalent join conditions

These are conditions involving two or more scope elements other than equivalent joins. These are done last because they are the most expensive in terms of processing cost. For example:

```
Customer.numOrders < AccountManager.threshold;
MyFunctions.match(Customer, AccountManager);
```

- If a rule uses multiple equivalent joins, a warning is printed to the engine log file. The purpose of the warning is to draw your attention to the situation so you can ensure that the order of the joins results in the most efficient processing.

- To optimize performance, do as much filtering as possible, to reduce the number of times TIBCO BusinessEvents evaluates a join condition.

- When using static variables, it is preferable to use Global Variables rather than scorecards for optimum performance. Scorecards are special types of concepts, which will create join conditions like any other concepts as opposed to Global Variables. Global Variales are a part of filter conditions, thus they yield greater performance.

Enforcing the Order of Condition Evaluation

To enforce the order of evaluation between two or more conditions, put them on the same line (that is, in one statement ending in a semicolon) joined by the logical operator &&.

Be aware of some differences in execution when you combine conditions. For example, consider the following separate conditions. A null pointer exception might be thrown if `concept.containedConcept` is null, if the second condition was checked before the first:

```
concept.containedConcept != null;
concept.containedConcept.property == "test";
```

You can, however, combine the conditions as follows:

```
concept.containedConcept != null && concept.containedConcept.property == "test";
```

In this case, a null pointer exception is not thrown when `concept.containedConcept` is null because it is always checked first.

Object Management (OM)

Object management refers to various ways that TIBCO BusinessEvents can manage the ontology object instances created by TIBCO BusinessEvents.

Cache object management enables rich functionality and is generally chosen for enterprise applications. In Memory object management can also play a useful secondary role in testing, and as an event router.

You can't mix different types of object management in one TIBCO BusinessEvents application. For example, you cannot deploy one engine using Cache OM and another using In Memory OM.

The goals of object management are as follows:

Object Persistence

Enables objects to be available for reuse, either in memory caches or in databases. Objects can also be recalled into the Rete network, thus extending the possible functionality of your system.

Data Recovery

Ability to survive failures without loss of data.

Object Partitioning

Ability to partition the objects among multiple JVMs. and to handle notifications of object additions, deletions, and changes to all the agents, enabling them to remain synchronized.

Object Clustering

The ability to maintain multiple copies of each object in different nodes (JVMs) such that if one node fails, another node can take over

Message Acknowledgement

For each message type (that is, each type of channel), TIBCO BusinessEvents acknowledges the receipt of messages according to the protocol of the messaging system.

Migrating to a Different Object Management Type

You can use In Memory object management in early phases of development. In later phases, you can implement Cache OM and take advantage of features it makes possible. Perform tests after changing object management type.

As with any change in configuration, be sure to perform thorough testing before going into production.

Summary of Object Management Features

The following table illustrates features supported for each OM type.

OM Type	Persistence	Data Recovery	Partitioning	Clustering	Fault Tolerance
In Memory	No	No	No	No	No (use Cache with Memory Only objects)
Cache	Yes	Yes	Yes	Yes	Yes (at agent level)

Cache Object Management

Object data is kept in memory caches using cache clustering technology, with redundant storage of each object for reliability and high availability.

Cache data is shared across all the engines participating in the cluster. Two cache providers are supported. The built-in cache provider is the TIBCO BusinessEvents DataGrid component. You also

have the option to use a supported version of Oracle Coherence, for which you have a license that is appropriate for your usage.

Cache Object Management Terminology

The following basic definitions apply:

Processing Unit

A processing unit deploys as a TIBCO BusinessEvents engine. One engine runs in one JVM.

Agent

Each processing unit contains one or more agents of different types. The main types are inference agents, which perform the inferencing work, and cache agents, which manage the objects.

Agent Class

An agent class is a configured agent definition. Configuration specifies, for example, what channels, startup rule functions, and rules the agent will use at runtime. You can deploy multiple instances of the same agent class, and you can deploy instances of different agent classes, depending on the work the application is designed to do.

Cache Agent

An agent that stores cache data. A processing unit can have one cache agent only. (Processing units that run other types of agents can have cache storage enabled too, which can be useful for demonstration purposes only, but not in production systems).

Seeders and Leeches (TIBCO BusinessEvents DataGrid terms)

A seeder is an agent that stores cache data. Cache agents are seeders by default. A leech is an agent that is part of the cluster but does not store data. Agents other than cache agents are generally leeches. These terms are not generally used in the documentation. However they may be useful to know about for technical discussions.

Data Recovery

Data recovery after total system failure is available if you implement a persistent backing store. Recovery from failure of individual processing units (JVMs) is available with Cache OM without a backing store (if at least one backup copy of each object is maintained in the cache).

Object management features provide fine-grained controls for managing the memory footprint of the cache, if you use a backing store.

Fault Tolerance

Fault tolerance is provided at the inference agent level. Agents belonging to the same agent class can act in a traditional fault-tolerant manner, where standby agents take over for failed active agents. Fault tolerance can also be provided implicitly, because all active agents in the same class share the workload. There may be no need to keep any agents as standbys. It depends on your needs.

Cache-based object management is generally the best choice for CEP systems. It offers richer functionality, and is the method that receives most focus in these chapters. For implementation details, see *TIBCO BusinessEvents Developer's Guid*e.

In Memory Object Management

In Memory object management does not persist object instances, which are maintained in local JVM memory only.

Objects are managed by standard JVM features. This is the only section on In Memory manager, because of its simplicity.

In Memory OM does not provide data recovery in case of system failure. The working memory on each system is not synchronized. Object state is not maintained. At startup after a failure, object state is initialized to the application's starting state.

The property `be.stats.enabled` allows you to turn on or off the aggregation of the metrics collected by TIBCO BusinessEvents, and in turn exposing them via JMX. It is honored only for in-memory mode and only if the property `com.tibco.be.metric.publish.enable` is true.

The In Memory option is a good choice for development and testing environments. In production environments, the In Memory option is best used for stateless operations and transient objects. An independently deployed In Memory application can act as an event router, directing events to agents in a cache cluster for processing.

For Fault Tolerance: In Memory OM itself does not support fault tolerance. If you require fault tolerance with an in memory system, then configure for Cache OM, but use the Memory Only mode for all objects. Because data is not persisted, it is lost during failover and failback. However, the engine process continues.

Another advantage of this approach is that the in memory processing units can participate in the larger cluster, instead of being a separately deployed application.

You can also control the performance of the statistics aggregators using the property `be.stats.threading.model`. The values are:

- `none` - use the calling thread for doing the aggregation (not recommended, but is useful in debugging).
- `single` - run all the aggregation in a single thread (default mode).
- `multi` - run the aggregations on multiple threads. There are four aggregators: destinations, events, engine, and thread pool. Each aggregator runs on an individual thread resulting in four new threads.

See In-Memory Performance Statistics Specifications for detailed specifications.

In Memory Performance Statistics Specifications

You can use different MBeans to gather statistics for different aggregators: destinations, events, engine, and thread pool.

All Destinations Statistics

Use the `[com.tibco.be.Agent.<AgentID>.Stats.Destinations.All]` MBean for statistics of all destinations.

Operation Name	Return Type	Notes
getStats(nameOrExpression)	- String name - long eventsReceivedPerSecond - String lastEventReceived - long totalEventsReceived - long totalEventsSent	Search for statistics using a full name or a regular expression. Use *<blank>* as argument to get all known destinations.

Operation Name	Return Type	Notes
getStatsByEventsReceived(nameOrExpression, boolean ascending)	• String name • long eventsReceivedPerSecond • String lastEventReceived • long totalEventsReceived • long totalEventsSent	Search for statistics using a full name or a regular expression sorted by events received (ascending or descending). Use *<blank>* as argument to get all known destinations.
getStatsByEventsSent(nameOrExpression, boolean ascending)	• String name • long eventsReceivedPerSecond • String lastEventReceived • long totalEventsReceived • long totalEventsSent	Search for statistics using a full name or a regular expression sorted by events sent (ascending or descending). Use *<blank>* as argument to get all known destinations.

Destination Statistics

Use the [com.tibco.be.Agent.*<AgentID>*.Stats.Destinations.*<Destination Name>*] MBean for statistics of the specified destination.

Operation Name	Return Type	Notes
getEventsReceivedPerSecond	long	The running events received per second (will be revised every time an event is received). If no event is received, then it shows the last computed value.
getLastEventReceived	long	Shows the time when the last "events received per second" was calculated. Effectively, it shows when the last event was received on a destination.
getTotalEventsReceived	long	The total number of events received after the destination became active.
getTotalEventsSent	long	The total number of events sent after the destination became active.

Engine Statistics

Use the [com.tibco.be.Agent.*<AgentID>*.Stats.Engine] MBean for statistics of the engine.

Operation Name	Return Type	Notes
getTotalRTCs	long	Total number of RTCs.
getAverageRTCTime	double	The average RTC time (in msecs).
getRulePerformanceStats(nameOrExpression)	• String uriOrSignature • long invocationCount • double averageProcessingTime • double averageConditionProcessingTime	Search for rule statistics using a full name or a regular expression. Use *<blank>* as argument to get all known rules/rule functions.
getRulePerformanceStatsByProcessingTime(nameOrExpression, ascending)	• String uriOrSignature • long invocationCount • double averageProcessingTime • double averageConditionProcessingTime	Search for statistics using a full name or a regular expression sorted by processing time (ascending or descending). Use *<blank>* as argument to get all known rules/rule functions.
getRulePerformanceStatsByConditionProcessingTime(nameOrExpression, ascending)	• String uriOrSignature • long invocationCount • double averageProcessingTime • double averageConditionProcessingTime	Search for statistics using a full name or a regular expression sorted by condition processing time (ascending or descending). Use *<blank>* as argument to get all known rules/rule functions.

Event Statistics

Use the [com.tibco.be.Agent.*<AgentID>*.Stats.Event] MBean for statistics of the event.

Operation Name	Return Type	Notes
getAverageEventProcessingTime	double	The average processing time per event.
getEventsPerSecond	long	The running events received per second (will be revised every time an event is processed). If no event is processed, then shows the last computed value.

Operation Name	Return Type	Notes
getLastEventProcessedTime	long	Shows the time when the last "events per second" was calculated. Effectively, it shows when the last event was processed
getTimerEventsFired	double	The number of timer events fired.
getTotalEventsReceived	long	The total number of events processed (includes timer events).

Scorecard Based Statistics

Use the `[com.tibco.be.Agent.<AgentID>.Stats.UserDefinedStats]` deployer MBean for scorecard based statistics.

Operation Name	Return Type	Notes
register (namepattern)	Integer (count of registered ScoreCards)	The MBean finds all the scorecards matching the name pattern and wraps them with a dynamic MBean. Each score card MBean is registered as `[com.tibco.be.Agent.<AgentID>.Stats.scorecard.<ScoreCardName>]`.
unregister (namepattern)	Integer (count of unregistered ScoreCards)	

All Thread Pool or Job Queue Statistics

Use the `[com.tibco.be.Agent.<AgentID>.Stats.tpool.All]` MBean for statistics of all thread pool.

Operation Name	Return Type	Notes
getStats(nameOrExpression)	- String name - long activeThreads - long maximumThreads - long queueCapacity - long queueSize	Search for statistics using a full name or a regular expression. Use *<blank>* as argument to get all known thread pools.

Operation Name	Return Type	Notes
getStatsByActiveThread(name OrExpression, boolean ascending)	• String name • long activeThreads • long maximumThreads • long queueCapacity • long queueSize	Search for statistics using a full name or a regular expression sorted by active thread count(ascending or descending). Use *<blank>* as argument to get all known thread pools.
getStatsByQueueSize(nameOrE xpression, boolean ascending)	• String name • long activeThreads • long maximumThreads • long queueCapacity • long queueSize	Search for statistics using a full name or a regular expression sorted by queue size(ascending or descending). Use *<blank>* as argument to get all known thread pools.

Thread Pool Statistics

Use the [com.tibco.be.Agent.*<AgentID>*.Stats.tpool_jqueue.*<Thread Pool Name>*] MBean for statistics of the specified thread pool.

Operation Name	Return Type	Notes
getActiveThreads	long	The total number of active threads.
getMaximumThreads	long	The maximum number of threads in the thread pool.
getQueueCapacity	long	The capacity of the job queue associated with the thread pool.
getQueueSize	long	The number of jobs in the queue associated with the thread pool.

Object Management and Fault Tolerance Scenarios

Fault tolerance and object management options work in various deployment scenarios to maintain data integrity.

The tables in this section help you understand how fault tolerance and object management options work and explain what is possible in each type of object management given the following conditions:

Processing Units (PUs)

One or multiple PUs, where a PU is a TIBCO BusinessEvents server running in one JVM.

Agents

One or multiple inference agents running in a PU. Each inference agent is configured by an agent class in the CDD. An inference agent has one or more Rete networks.

When implementing a recovery strategy you must take care to maintain the integrity of stateful objects. Concepts and scorecards are stateful objects and must maintain state across inference agents.

Cache OM with Memory Only Mode on All Objects and Fault Tolerance

In Memory object management does not support fault tolerance.

This table presents options available if you use Cache OM with Memory Only mode set on all objects, which provides fault tolerance for memory only objects.

Cache OM with Memory Only Mode on All Objects and Fault Tolerance Scenarios

# PUs #Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
1 PU 1 Agent	(N/A)	Data is isolated to a single PU (JVM). No recovery.
1 PU *n* Agents	(N/A)	No recovery.
n PUs 1 Agent	Data is isolated in each PU. Failover and failback are allowed. Object state is not preserved or transferred. Recommended only for stateless operations.	Data is isolated to each PU. No recovery.
n PUs *n* Agents	Data is isolated in each multi-agent PU. Object state is not maintained during failover and failback. Recommended only for stateless operations.	No recovery.

Cache OM and Fault Tolerance

Fault tolerance of the engine process refers only to inference agents.

In all cases it is assumed that dedicated cache agents are also running.

If you use multi-engine (multi-agent) features, fault tolerance is implicit. When all agents in an agent group (an agent group consists of instances of the same agent class) are active, if any active agent fails, remaining agents in the group automatically handle the workload.

In all cases, in the event of total system failure, use of a backing store ensures recovery of data written to the backing store.

Cache and Fault Tolerance Scenarios

# PUs #Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
1 PU 1 Agent	(N/A)	(N/A)

# PUs # Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
1 PU *n* Agents	(N/A) Each agent in the same PU is a different agent, not part of the same agent group.	(N/A)
n PUs 1 Agent	Fault tolerance is at the agent level. If one or more agents in a group fails, the load is distributed among remaining agents in that group. All agents can be active or some can be standbys. Configuration uses a `MaxActive` property and a `Priority` property. Cluster data is shared between agents across all PUs, using the cache cluster. If the number of cache object backups is one, one cache agent (at a time) can fail with no data loss. With two backups, two servers can fail, and so on. Caches exist in memory only, so recovery is not available in the case of a total system failure. In the event of total system failure, use of a backing store ensures recovery of data written to the backing store.	N/A. Fault tolerance is implicit.
n PUs *n* Agents	Same as *n* PUs 1 agent. Each of the agents in one PU is fault tolerant with the agents in the same agent group, which are deployed in other PUs.	Multi-agent mode: N/A. Fault tolerance is implicit.

Distributed Cache OM

Cache object management (OM) is the standard choice for most TIBCO BusinessEvents Cache Object Management Feature Overview.

Cache-based object management is generally the best choice for a CEP system, and a distributed cache is generally the most appropriate, especially when used with a backing store (database). All the provided caching schemes use a distributed cache and are configured for production as shipped.

Cache OM is a requirement for certain features such as multi-agent concurrency.

Object management is configured using the Cluster Deployment Descriptor (CDD), an XML file that you edit in TIBCO BusinessEvents Studio using a provided editor. For more information, see *TIBCO BusinessEvents Configuration Guide.*

Distributed Cache Characteristics

In a distributed cache, cached object data is partitioned between the PUs (JVMs) in the cache cluster for efficient use of memory. By default one backup of each item of data is maintained, on a different PU. You can configure more backups of each object to be kept on different PUs to provide more reliability as desired, or to disable maintenance of backups.

Distributed caching offers a good balance between memory management, performance, high availability and reliability.

Scaling the System

Scaling is linear. To scale the system's capacity to process more data, add more inference agents. To scale the cache, add more cache servers .

In addition, each entity can have a different cache mode, to help you balance memory usage and performance .

Reliability of Cache Object Management

When you use Cache object management without a backing store, objects are persisted in memory only, and reliability comes from maintaining backup copies of cached objects in memory caches.

To provide increased reliability in the case of a total system failure, add a backing store.

Multi-Agent Concurrency Features

Multiple inference agents can run concurrently in either of two ways. In both cases the agents share the same ontology and same cache cluster:

- Multiple instances of the same inference agent class, each running on different PUs, form an *agent group*. This provides simple load balancing of messages arriving from queues, as well as fault tolerance. You can also configure content-aware load balancing for "session stickiness." (See Load Balancer Configuration in *TIBCO BusinessEvents Developer's Guide*.)

- Different agents in different PUs work concurrently to distribute the load on the JVM processes. This results in quicker conflict resolution and the ability to handle a heavy incoming message load. For example, Agent X connects to Agents Y and Z to create rule chaining across a set of PUs. Each agent uses different sets of rules, such as rules for fraud, upsell and cross-sell. All agents operate against the same cluster and share the same ontology. The output from one agent may trigger rules deployed in another agent, causing forward chaining of the workload.

Concurrent RTC: You can also enable concurrency within a single agent, using the multi-threaded Rete feature, known as concurrent RTC (and in prior releases as `concurrentwm`). Within one agent, multiple RTC cycles take place concurrently.

Concurrent RTC does not require cache.

Concurrency and Locking

With agent concurrency and concurrent RTC features, you must use locking: in both cases multiple RTCs are being processed at the same time, and data must be protected as in any concurrent system.

Characteristics of a Distributed Caching Scheme

The cache characteristics are defined by a caching scheme.

TIBCO BusinessEvents uses a distributed caching scheme, in which the cached object data is partitioned between the storage PUs in the cache cluster for efficient use of memory. This means that no two storage PUs are responsible for the same item of data.

A distributed caching scheme has the following characteristics:

- Data is written to the cache and to one backup on a different JVM (replication count can be set to none, one, or more backup copies, depending on configuration). Therefore, memory usage and write performance are better than in a replicated cache scheme. There is a slight performance penalty because modifications to the cache are not considered complete until all backups have acknowledged receipt of the modification. The benefit is that data consistency is assured.

    Each piece of data is managed by only one cluster node, so data access over the network is a "single-hop" operation. This type of access is extremely scalable, because it can use point-to-point communication and take advantage of a switched network.

- Read access is slightly affected because data is not local. The cache is distributed between the cache agent nodes.

- Data is distributed evenly across the JVMs, so the responsibility for managing the data is automatically load-balanced across the cluster. The physical location of each cache is transparent to services (so, for example, API developers do not need to be concerned about cache location).

- You can add more cache agents as needed for easy scaling.

- The system can scale in a linear manner. No two servers (JVMs) are responsible for the same piece of cached data, so the size of the cache and the processing power associated with the management of the cache can grow linearly as the cluster grows.

Overall, the distributed cache system is the best option for systems with a large data footprint in memory.

Failover and Failback of Distributed Cache Data

The object manager handles failover of the cache data on a failed cache agent and it handles failback when the agent recovers.

It is not necessary to use fault tolerance for cache agents: the cluster transparently handles failover of data to other cache agents if one cache agent fails.

When a node hosting a cache agent fails the object manager redistributes objects among the remaining cache agents, using backup copies, if the remaining number of cache agents are sufficient to provide the number of backups, and if they have sufficient memory to handle the additional load. However, because this is a memory-based system, if one cache agent fails, and then another cache agent fails before the data can be redistributed, data may be lost. To avoid this issue, use a backing store.

If redistribution is successful, the complete cache of all objects, plus the specified number of backups, is restored. When the failed node starts again, the object management layer again redistributes cache data.

Specifically, when a cache agent JVM fails, the cache agent that maintains the backup of the failed JVM's cache data objects takes over primary responsibility for that data. If two backup copies are specified, then the cache agent responsible for the second backup copy is promoted to primary backup. Additional backup copies are made according to the configuration requirements. When a new cache agent comes up, data is again redistributed across the cluster to make use of this new cache agent.

Because they store data in memory, cache-based systems are reliable only to the extent that enough cache agents with sufficient memory are available to hold the objects. If one cache agent fails, objects are redistributed to the remaining cache agents, if they have enough memory. You can safely say that if backup count is one, then one cache agent can fail without risk of data loss. In the case of a total system failure, however, the cache is lost.

Limited and Unlimited Cache Size

Performance of the system is best when all the data is in cache, which can have unlimited or limited size.

If the amount of data exceeds the amount of memory available in the cache machines, you must limit the cache size and use a backing store to store additional data. Some applications use the backing store as the main storage and retrieve objects from the backing store as needed.

Use of limited cache is supported only with the use of a backing store, which retains entries in excess of the limit. Without use of a backing store the following data inconsistencies could result:

- Entries for an object in the object table (an internally used cache) and in the object cache itself could expire independently of each other.

- Domain object settings for limited cache apply at the object level. Related concepts could have different settings. For example, a container concept could have a limited cache setting and its container concept an unlimited cache setting. Each could be evicted at different times.

With a limited cache, objects are evicted from the cache when the number of entries exceeds the limit. The Coherence cache provider uses a hybrid policy. TIBCO BusinessEvents DataGrid uses a Least Recently Used (LRU) policy.

A hybrid eviction policy chooses which entries to evict based on the combination (weighted score) of how often and how recently they were accessed, evicting first those that are accessed least frequently and have not been accessed for the longest time.

The evicted objects are transparently loaded from the backing store when needed by agents.

Only use an unlimited cache if you deploy enough cache agents to handle the data. Otherwise out of memory errors may occur.

For backing store configuration, see JDBC Backing Store Setup in *TIBCO BusinessEvents Developer's Guide*.

Distributed Cache and Multi-Agent Architecture

Different TIBCO BusinessEvents processing units and agents within the processing units have specialized roles in a Cache OM architecture.

The drawing below illustrates Cache OM architecture.

Cache Object Management and Fault Tolerance Architecture

The drawing illustrates one possible configuration, and assumes destinations that are JMS queues using basic load balancing (Content-aware load balancing is also available. See Load Balancing).

Agent group IA1 has three active agents, and agent group IA2 has two load balanced agents and one standby agent for fault tolerance. Each agent group is listening on a different destination.

Agent classes and processing units can be configured at deploy time (within the constraints of the project).

For more information about designing a project that uses multiple agents, see the following sections:

- Load Balancing
- Fault Tolerance of Agents

Cache Clusters (Metaspaces)

A cache cluster (or *metaspace,* to use the TIBCO BusinessEvents DataGrid term) is a logical entity that provides the following services:

- Cache Management: Partitioning, replication, distribution and failure recovery (see Reliability of Cache Object Management).
- Fault Tolerance (of data): Notifications to inference agents so that the state of each agent's working memory remains synchronized with the others, so any agent in the cluster can take over in event of a JVM failure.

You define the cluster member machines, processing units, and agents in the Cluster Deployment Descriptor (CDD) which is an XML file, configured in the CDD editor in TIBCO BusinessEvents Studio. See Cluster Deployment Descriptor (CDD) in *TIBCO BusinessEvents Configuration Guide*.

Processing Units

Each processing unit in a cache cluster runs in an instance of a Java virtual machine (JVM). It hosts one or more TIBCO BusinessEvents agents. Each processing unit with storage enabled participates in the distributed cache.

Processing units with inference agents also have an L1 Cache, a local cache that gives inference agents quick access to recently used objects.

Agents

There are several types of agents: *inference agents*, *cache agents*, *query agents*, and *dashboard agents*.

Inference Agents

An i*nference agent* executes rules according to the rule agenda created using the Rete network.

In Cache OM systems, inference agents are connected to the cache cluster, enabling fault tolerance of engine processes and cache data, as well as load balancing (with queues).

At design time, you configure an *inference agent class* with a selection of rules from the project, and a selection of destinations, and, as needed, a selection of shutdown and startup functions.

Cache Agents

The purpose of *cache agents* is to store and serve cache data for the cluster.

The built-in cache provider, TIBCO BusinessEvents DataGrid, calls such agents *seeders*. The Oracle Coherence cache provider calls them *storage nodes*.

Dedicated cache agent PUs are non-reasoning agents (one per PU). Cache agents are responsible for object management. They participate in distribution, partitioning and storage of the objects in the cluster.

Other agent nodes functioning as cache agents. It is possible, but not recommended, to enable inference and query agent nodes to act as seeders (storage nodes) in addition to their other functions. Using dedicated cache agent nodes for data storage is more efficient and more scalable for production scenarios. Enabling storage on a different kind of agent can be convenient during testing.

When a backing store is used, you can balance what objects to keep in the cache and what to keep in the backing store, until needed.

Memory and Heap Size Guideline for Cache Agents

Guidelines for memory and heap size depend on the cache provider.

With TIBCO BusinessEvents DataGrid Cache Provider

The JVM can use much more memory than expected because it often defers garbage collection. If the JVM uses a large percentage of the available physical memory, TIBCO BusinessEvents DataGrid might not perform well, due to swapping. The JVM and TIBCO BusinessEvents DataGrid run in the same process; therefore they compete for the same addressable space in the RAM. If the JVM uses a large percentage of the addressable space, out of memory errors can occur in TIBCO BusinessEvents DataGrid. This situation is more likely to occur on 32-bit systems, where the addressable space is 4GB or less. To avoid this situation, set a heap limit to restrict the amount of memory used by the JVM, so that it does not compete with TIBCO BusinessEvents DataGrid. For example, you could use the command line option -Xmx512m.

With Oracle Coherence Cache Provider

With Oracle Coherence as the cache provider, the amount of memory you need for cache agents depends on factors such as how many objects you have, their object management configuration, and whether you are using limited or unlimited cache.

You must find an appropriate balance for your projects between too little memory, which leads to too much time spent in garbage collection, and too much memory, which leads to longer garbage collection cycles. The optimal heap size depends on factors such as how much data is kept in each cache agent, how many cache agents are used, and whether the cache is limited or unlimited.

For example, if you use a JVM heap size of 1024 MB (1GB), in order to minimize the impact of garbage collection, about 75% of the heap can be used to store cache items, which means about 768 MB per heap. The other 25% is then available for garbage collection activities.

Different operating systems have different requirements: adjust as required.

Query Agents

Query agents are available only with TIBCO BusinessEvents Event Stream Processing add-on software.

Query agents use an SQL-like query language. You can query data that is in the cache. You can also query data arriving in events, known as event stream processing or ESP.

A query agent is a non-reasoning agent. It has read-only access to the underlying objects in the cache cluster. A query agent can execute rule functions, but not rules. You can mix query agents and inference agents within one node as desired.

TIBCO BusinessEvents Event Stream Processing Query Developer's Guide explains how to work with the query language.

Dashboard Agents

Dashboard agents are available only with TIBCO BusinessEvents Views add-on software.

Dashboard agents are similar to a query agent in that their role is to generate information based on queries. The information is made available to the TIBCO BusinessEvents Views dashboard.

See *TIBCO BusinessEvents Views Developer's Guide* for details.

Cache Cluster Member Discovery

There are two methods you can use to configure how the members of the cache cluster are discovered: multicast and well-known addresses.

By default, multicast is used, and in many cases default multicast values work without additional configuration.

For more information on cluster configuration, see *TIBCO BusinessEvents Configuration Guide*.

Cluster Member Discovery Using Multicast Discovery

If multicast is used, the cluster membership is established using the multicast IP address and port. When a TIBCO BusinessEvents processing unit (node) subscribes to this multicast IP address it broadcasts information about its presence to the address.

Multicast is used to discover new processing units (nodes) and add them to the cache cluster. Similarly when nodes are removed or moved to a different server, the multicast protocol ensures that members are kept current without any additional configuration.

Default values provided mean you may not have to configure any discovery-related properties. However, if you deploy multiple TIBCO BusinessEvents projects in your environment, you must specify different multicast address and port settings for each project.

Cluster Member Discovery Using Well-Known Addresses

When multicast is undesirable or unavailable, for example, if nodes are deployed to different subnets and broadcast between the subnets is not enabled, use well-known addresses (WKA) instead.

To use well-known-addresses you provide the IP addresses and ports of certain members to all potential, using CDD settings.

The disadvantage of well-known addresses is that configuration is somewhat fixed, and at least one of the well-known-address members must be running at all times so that new members can join the cluster.

Load Balancing

Load balancing is available for messages arriving from queues. Do not use load balancing for subject-based or other broadcast sources.

Two kinds of load balancing configuration are available: basic load balancing and content-aware load balancing. They support messages arriving from TIBCO Enterprise Message Service queue sources.

Every JMS destination that is configured to be an input destination runs in its own JMS Session. This provides good throughput on queues and topics for processing, and less connections.

Basic Load Balancing

Events from queue sources are automatically distributed between instances of an agent class. To set up this kind of load balancing, you simply deploy multiple instances of an agent class, where each agent runs in a different processing unit.

Certain aspects of the design have to be managed by the application. This method can be useful when there is no relationship between the events that would require them to be processed in a certain order. If the order or grouping of events received is important, use content-aware load balancing. Content-aware load balancing has other benefits also, as explained below.

Content-aware Load Balancing

With content-aware load balancing, all related events arriving from queues are routed to the same agent. The events arriving at a destination are related by a routing key, which uses the value of a selected event property. For example, if the event property values are zip codes, then all messages relating to one zip code are routed (over TCP) to the same receiver agent, providing "session stickiness."

Use of content-aware load balancing simplifies project configuration, and makes runtime behavior more efficient. For example, only local locking is generally required (whereas basic load balancing requires cluster-wide locking). Also the L1 cache does not have to be checked for version consistency.

Fault Tolerance of Agents

Inference and query agents in an agent group (that is, all agent instances of the same agent class deployed in the same cluster) automatically behave in a fault tolerant manner.

Cache agents do not need or use fault tolerance features. Fault tolerance of cache agents is handled transparently by the object management layer. For fault tolerance of cache data, the only configuration task is to define the number of backups you want to keep, and to provide sufficient storage capacity. Use of a backing store is recommended for better reliability (see Reliability of Cache Object Management).

All load is distributed equally within all active agents in the same group. If any agents fail, the other agents automatically distribute the load between the remaining active agents in the group.

You can optionally start a certain number of agents in a group and keep the rest as standby agents. If an active agents fails, a standby agent is automatically activated. For most situations, however, there is no need to maintain standby agents.

Fault Tolerance Limitation in Inference Agents: Entities that use Memory Only cache mode are not recoverable in failover or failback situations.

Behavior of Standby Agents

Query agents do not maintain stateful objects. When a standby agent becomes active, it simply begins to take on work.

Standby inference agents maintain a passive Rete network. They do not listen to events from channels, do not update working memory, and do not do read or write operations on the cache.

Startup rule functions do not execute on failover: When a standby or inactive node becomes active, it does not execute startup rule functions.

Cache OM with a Backing Store

To provide data persistence, you can implement a backing store for use with Cache OM.

During regular operation, cache data is written to the backing store. On system restart, data in the backing store is restored to the cache cluster.

Database Disconnection Situations: In the event of a lengthy database disconnection, no events are processed until the database connection is restored. Depending on configuration, event processing may continue for some time to avoid disruption from short disconnections. For example, engine processes are blocked only when the database write queue is full (either the write-behind queue or the cache-aside queue). For more on these options see Database Write Tuning Options for Cache Aside .

Implementing a Backing Store

To implement a backing store, you provide a supported database product. Scripts are provided to set up the database for your project's ontology. If the ontology changes, scripts help you adapt the backing store accordingly (though some manual work may be required depending on the nature of the changes). Existing backing store data can be preserved.

For backing store configuration, see JDBC Backing Store Setup in *TIBCO BusinessEvents Configuration Guide*.

Configuring Backing Store Options

Various options are available for configuring the backing store for your needs, as explained in this chapter. See Storage and Retrieval of Entity Objects for more information about fine-grained controls over data storage and retrieval.

Backing Store Write Options — Cache-aside and Write-behind

Writes to the backing store can be done in either of two ways: *cache-aside*, in which the inference agent handles all writes simultaneously, and offers transaction control; or *write-behind*, in which the cache agent handles writes to cache then database.

For most systems, especially those with a lot of I/O, including updates and deletes within an RTC, cache-aside is recommended.

Cache-aside

With cache-aside database write strategy, inference agents manage writes to the cache, the local L1 cache, and the backing store, simultaneously, in the post RTC phase. (The cache agent reads from the backing store, but does not write to it.)

Cache-aside allows batching of writes to the backing store and provides thread and queue size controls.

Write-behind

With write-behind database write strategy, the cache management layer handles writes to cache and to the backing store. First it writes data to the cache and then to the backing store. Writes are managed by the cache agents. For inserts and updates, one write-behind thread is used for each entity type. Deletes are performed by the distributed cache threads (configurable) and are synchronously deleted from the database.

Write operations from multiple writers to the cache are batched. Write-behind batches the writes during the delay period which increases database call efficiency and minimizes network traffic.

Write-behind does not offer transaction controls, and can be slower than cache-aside.

If enough cache agents fail, the cache management layer won't be able to persist a write that was done previously, resulting in an inconsistent database.

Starting a Minimum Number (Quorum) of Cache Agents

At system startup, one node in the cache cluster loads objects from the backing store to the cache cluster, according to the preloading settings (see Storage and Retrieval of Entity Objects). Any node in the cluster can perform the preloading.

Before preloading begins, you must ensure that enough cache agents have started to hold the objects from the backing store. The cluster does not start processing incoming data until the required objects have been loaded into the cache.

See Cache OM and Cluster Configuration in *TIBCO BusinessEvents Configuration Guide*, for details on specifying the minimum number of cache agents that must start before cache loading begins.

After the specified number of cache agents has started, the processing unit that acquires the lock first performs the cache loading. Any processing unit can acquire the lock. All agents wait until backing store data has finished loading before they start.

> This setting does not affect runtime operation of the deployed application. Deployed applications continue to run even if one or more cache agents fails and the quorum is no longer met. A warning message is written to the log file.

Storage and Retrieval of Entity Objects

When you use Cache OM and a backing store, various options help you manage where entity objects are stored, and how to retrieve them from the backing store at startup to optimize system performance and memory management.

With Cache OM, objects created by a running TIBCO BusinessEvents application can be kept in any of these locations:

- The Rete network (JVM memory)
- The cache
- The backing store

You can manage where the object data is kept at the level of the entity type. The best choice depends on how often the object changes, and how often it is accessed. The various options balance the memory and performance characteristics of the system. Different applications have different priorities and it is up to you to choose the options that suit your needs.

Between Backing Store and Cache Preloading Options and Limited Cache Size

Best performance is obtained when all objects are in the cache, but in practice there are often more objects than you can or want to keep in the cache.

When the system demands an object that exists in backing store but not in cache, the object is automatically loaded from the backing store into the cache, and then into the Rete network. This takes time, but reduces the need to store so much data in the cache, which uses up memory.

You can configure what objects to preload into cache on startup, and what objects to evict from the cache when not needed. You can preload all, none, or entities of selected types. You can also configure what object handles to preload into the *object table*. Again, you can preload handles for all, none, or selected types. The first RTC does not occur until the object table has been preloaded (with all the object handles configured for preloading). For more details, see The Role of the Object Table.

It is also important to start enough cache agents to handle the work.

Limiting Cache Size

When you use a backing store, you can limit the size of the cache by specifying the cache size. This is helpful for large volumes of data that the available memory cannot hold. When the number of objects in cache reaches the cache size limit, some of the objects are automatically evicted from cache (they are still in the backing store).

See Cache OM and Cluster Configuration in *TIBCO BusinessEvents Configuration Guide*.

Between Cache and Rete Network Cache Modes

Less frequently used objects can be stored only in the backing store, and retrieved into the cache as needed.

Cache Plus Memory (Cache+Memory) mode is deprecated.

In a similar way, you can define how to manage the instances of each object type, using one of the following *cache modes*:

Cache Plus Memory

Shown as "Cache+Memory" in the UI.

In the Rete network as well as in the cache: Use for constants and concepts that change infrequently. In a concurrent system, Cache Plus Memory mode is inappropriate for most purposes, due to the difficulties involved in keeping all the concurrent processes synchronized.

Cache Only

Only in the cache, most commonly used mode for concurrent systems. The objects are retrieved into memory (the Rete network) only when needed for an RTC. Locking is still required as in any concurrent system.

Memory Only

Only in the Rete network, used for objects whose persistence is not important. Use for small, frequently used, stateless entities, for improved performance.
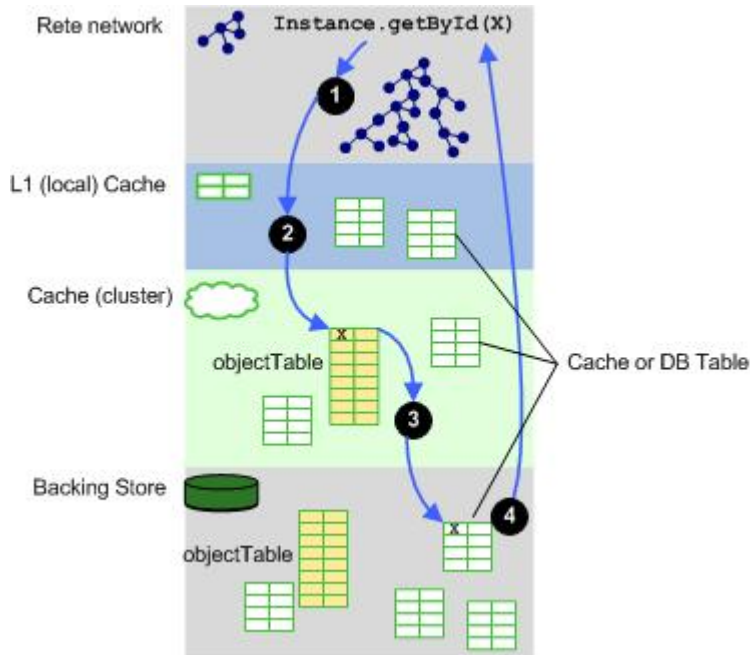
The Role of the Object Table

Preloading controls are available for entity objects, and for entries (handles) in the object table (`objectTable`) cache relating to entity objects.

The `objectTable` cache is a large cache that provides mappings for all entities in the cache. The object table contains the object id and information about the object such as its extId, class name, type, backing store table name and cache name. The object table is used to find the actual object either in the cache or in the backing store. The object table is also written to the backing store.

The object table can grow become very large, up to hundreds of millions of rows in extreme cases. You can preload the cache cluster's object table at system startup to improve performance after the system has started up. (See Domain Objects Configuration in *TIBCO BusinessEvents Configuration Guide* for details.)

The following figure shows how the object table in cache is used at runtime.

Use of ObjectTable at Runtime

1. A rule requests an object X to be fetched using its internal ID. The object is not in the Rete network.

2. The object is not in the local cache.

3. TIBCO BusinessEvents looks up the ID in the object table and gets its cache name. Its cache table is not found in the cache cluster.

4. TIBCO BusinessEvents gets the name of the object's backing store table from the object table, locates the object's table in the backing store, and returns the object to the calling function.

If the object table is not preloaded with the entry for object X, then an additional processing step is required, to load the object table in the cache for that object's entry.

Cache Modes and Project Design

The Rete network consumes a large amount of the available memory in the JVM.

You can use cache modes to tune the performance of your application, and reduce its footprint in memory. You can keep memory objects in the cache or Rete network using the following cache modes:

Cache Plus Memory

In the Rete network as well as in the cache.

Cache Only

Only in the cache.

Memory Only

Only in the Rete network, depending on your need.

This section describes the cache modes available in more detail, and explains how to use them appropriately. For configuration details, see *TIBCO BusinessEvents Configuration Guide*.

Cache Modes are Set on Individual Entities to Tune Performance

You set cache modes at the level of individual entity types in your project.

This fine granularity allows you to tune performance and memory usage based on the size and usage of the concepts, scorecards, and events in your project ontology.

For example, you can use the *memory only* mode so that frequently used stateless entities are kept in memory (and are not cached). Objects kept in memory are highly available to the application.

Using Cache Only mode reduces the memory footprint. You must explicitly load the objects (in rules or rule functions) so they are available to the Rete network.

Do not mix *memory only* with *cache modes* in related concepts!

All concepts related by hierarchy, containment, or reference must use either:

⚠
- *memory only* mode
- *cache only* mode or *cache plus memory* mode, or both (but not *memory only*).

See Concept Relationships for more details about the relationships between concepts.

Cache Plus Memory — For Constants and Less Changeable Objects

With the *cache plus memory* setting, the entity objects are serialized and are always available in cache.

There is one object in the cache (in a logical sense), and any number of references (handles) to that object in each JVM. References to the entities are tracked in the working memory so they can be deserialized and retrieved from the cache when needed by the engine.

The agent's working memory is used to store the Rete network for the loaded rules. The inference agent does not store the actual object. It relies on the object management layer to load the objects on demand from the backing store. For example, consider a rule of this form:

```
Declaration (Scope): Customer
```

```
Condition: Customer.age < 20
```

If the cache mode is *cache plus memory*, then working memory stores handles to all customers, including those whose age is greater than 20. The customer objects for customers whose age is less than 20 are deserialized and retrieved from cache when the rule condition is evaluated, in order to process the rule.

Because a Rete network is so large, the references themselves can consume large amounts of memory. To reduce the memory footprint, it is recommended that you use the Cache Only mode for most entity types.

In addition, the `Cache Only` mode is more straightforward to use as there is less to synchronize in a multi-agent environment when the Rete network is flushed after each RTC.

Appropriate locking in complex projects that use concurrency features can be difficult with *cache plus memory*.

Memory Only — Useful for Stateless Entities

When you select *memory only* mode for an entity type, instances of that entity are available only in the engine's local JVM memory only.

These entities and their properties are not recoverable, or clustered or shared. For this reason, it is recommended that you use this mode for stateless entities only.

Memory only mode is typically used for static reference data that can be created in the rule functions on startup. It can be used also for transient utility entities that created and deleted within a single processing, and are not needed across RTC cycles.

Entities configured in *memory only* mode do not persist objects to the cluster and correspondingly the objects are not recovered from the cluster.

This cache mode works the same as the In Memory object management option (but is set for individual objects).

Setting scorecards to *memory only* mode: In a cache-based project, setting scorecards to *memory only* mode can improve performance.

Fault tolerance limitation: Entities that use *memory only* cache mode are not recoverable in fault tolerance failover or failback situations.

Cache Only Mode

As with the *cache plus memory* mode, when you choose the *cache only* mode for selected entities, the entity objects are serialized and are always available in cache.

The difference is that at the end of the RTC, the objects and their references are removed from the Rete network, thus freeing memory

The *cache only* mode is stateless between RTCs. You must explicitly load the objects needed by rules in an RTC, and you must ensure proper locking is used.

Various functions are available for loading the entities into the Rete network. They are generally used in an event preprocessor.

Cache Only Objects in the Rete Network

When you use *cache only* mode for an entity type, objects of that type behave normally when they are created during an RTC (see Conflict Resolution and Run to Completion Cycles for more details). At the end of an RTC, however, they are removed from the Rete network and written to the cache.

Ensure you use correct locking before loading objects.

Cache Load Functions

You must explicitly load Cache Only objects into the Rete network when they will be needed during an RTC, using an appropriate cache load function in the `DataGrid` function category, within the `Cluster` category:

```
CacheLoadConceptByExtId()
CacheLoadConceptsByExtId()
CacheLoadConceptById()
CacheLoadEventByExtId()
```

```
CacheLoadEventById()
CacheLoadParent()
```

Less-commonly used:

```
CacheLoadConceptByExtIdByUri()
CacheLoadConceptIndexedByExtId()
CacheLoadEntity()
cacheLoadEventByExtIdByUri()
loadConceptUsingExtId()
```

The functions that load concepts by `ExtID` or `ID` have a parameter to control whether contained concepts are also loaded. The `CacheLoadParent()` function, which loads a given concept's parents, has the option to return all parents or the immediate parent. (Parents are concepts related to the given concept by a containment relationship).

Referenced Concepts:

- If the referenced concept will be used in a rule agenda in working memory, you must explicitly load all referenced concepts as needed since only containment relationships can be handled automatically.

- If the referenced concept will *not* be used in a rule agenda in working memory and only the concept's property will be referenced, then it is not necessary to explicitly load the concept.

A general best practice is to use these functions in an event preprocessor. Event preprocessors are multithreaded so performance is better. Also, if you load the objects in the preprocessor, then the rules themselves do not have to take account of the need to load the objects during execution. For example, in the preprocessor, you could preload an order concept using an `ExtID` available in the event as follows:

```
Concepts.Order order =
Cluster.DataGrid.CacheLoadConceptByExtId(orderevent.Order_Id, false);
```

Loaded Objects Are Not New and Do Not Trigger Rules to Fire

Loaded objects do not behave like newly arrived entities.

The loaded objects are not asserted: their presence alone does not trigger rules. They are simply restored to the Rete network. They behave as if they had never been removed. For example, rules do fire if there is a join condition between the entity loaded from cache and another entity that is asserted or modified in the same RTC.

Also if you modify the object that you reloaded, it can trigger the rule.

Limited Use of getByExtId(): Only use this function to retrieve cache only objects that have already been loaded into the Rete network by a preprocessor. The getByExtId() function does not load the object into the Rete network.

Concurrency and Project Design

You can use multiple concurrently active inference agents to achieve load balancing, scaling, and performance.

You can also enable concurrent RTC cycles within one agent, known as the *concurrent RTC* feature.

The number of possible concurrent RTCs is determined by the number of available threads.

Both multi-agent and concurrent RTC features provide concurrent RTC functionality — across agents in the case of multiple agents, and within agents, in the case of concurrent RTC. As with any concurrent system, care must be taken to ensure that two agents or RTCs do not attempt to update the same instance at the same time, and to ensure that reads return a valid and up-to-date instance of an object.

Multi-Agent Features and Constraints

Concurrency affects the way events and objects are processed in a multi-agent configuration, or with concurrent RTC.

Multi-agent features can be used in two ways:

* Deployment of instances of the same agent, each in a different processing unit, for load balancing and fault tolerance. (A processing unit is one JVM, also known as a node.)

* Deployment of instances of different agents, to achieve rule-chaining for high distribution of the workload and high performance.

In both multi-agent cases, the agents are in the same cache cluster and work on the same ontology objects and, to provide performant systems.

Concepts are Shared Across Agents Asynchronously

All concepts are shared between agents in the cluster in an asynchronous manner. For instance, an Agent X receives an event E, fires a rule R1 that creates a concept C1. An agent Z receives an event E2, fires a rule R2 that joins concept C1 and event E2.

Therefore, there is inherent latency between an object change in an agent, and reception of the notification by the other agents in the cluster.

Because of the asynchronous sharing of objects between agents, ensure that events have a long enough time to live setting so that they do not expire before all actions pertaining to the event are done.

It is recommended that you explicitly consume events when their work is done so that they do not cause any unwanted (unforeseen) actions to occur by their presence in the Rete network.

Scorecards are Local to the Agent

Scorecards are not shared between agents. (This is true in all OM types.) Each inference agent maintains its own set of scorecards and the values in each agent can differ. This enables scorecards to be used for local purposes and minimizes contention between the agents.

Any agent that uses scorecards, and also uses Cache OM, must be assigned a unique key so that the correct scorecard can be retrieved from the cache. The key is set in the Processing Unit tab of the CDD.

By default the key is empty. When you start multiple InferenceAgents with same key, the scorecard will be shared by those agents.

As an analogy consider a bank ATM scenario. Money can be drawn from the same account using different ATMs. However, each ATM maintains a "scorecard" indicating only how much money it dispenses.

An agent key property (`Agent.AgentClassName.key`) is available for tracking scorecards. It identifies an agent uniquely so that its scorecard can be restored from the cache.

> Do not use scorecards as a mechanism to share data between multiple agents. Consider using concepts instead.

Events are Owned by the Agent that Receives Them

In a load balanced group of agents, events (messages) received by an agent are owned by that agent. Even when the event is retrieved from cache (for example for a join condition), the ownership is maintained. No other agent can work with that event, unless the owning agent fails.

In the event of engine failure, cache cluster services provide for reassignment of ownership of clustered events to other agents in the same agent group, so there is no single point of failure. (Of course, if the event's time-to-live period expires during this transition, the event expires and is not reassigned.)

Events from Queues

Events received from a queue are each taken up by one agent or another. For example, when an agent X receives an Event E1 from a queue, agent B in the same agent group does not see the event. To set up load balancing of events arriving from a queue, you enable the same destination in the agents you want to use. Agents used for load balancing are generally agents in the same group (class) but do not have to be.

Events from Topics

Unlike messages received from a queue, messages sent on a topic are received by all agents that actively listen to the topic. Each agent generates its own event instance (with its own ID) when receiving the message. While it could be useful for multiple agents to receive events sent on a topic, this often leads to undesirable results. Care must be taken to ensure that just one agent receives topic-based messages.

Event-Related Constraints

There are some concurrency constraints, which are related to particular events.

Constraints are as follows:

Repeating Time Events Not Supported

Time events configured to repeat at intervals are not supported in multi-agent configurations. Rule-based time events, however, are supported.

State Machine Timeouts

State machines can be configured to have state timeouts. The agents in the cluster collaborate to take ownership of management of the state machines, thereby providing automatic fault tolerance. (Requires TIBCO BusinessEvents Data Modeling add-on software.)

Multi-Agent Example Showing Event Handling

In a load balancing group shown in the example, concepts are shared and events are not shared .

Example Scenario

Agent A has the following rules:

Rule	Scope	Condition	Action
R1	Event E1	None	Create concept C1
R2	Event E2, Concept C1	E2.x == C1.x;	Send event E3

Agent A listens to destinations on which events E1 and E2 arrive.

You start two instances of agent A called A1 and A2. Both are active, therefore they both listen to the destinations on which events E1 and E2 arrive. At runtime, the arrival of two events illustrates the expected behavior.

Agent A1 receives an instance of Event E1:

1. Rule R1 executes and creates an instance of concept C1.

2. During the post RTC phase, the instance of C1 is written to cache.

3. Event E1 has a Time to Live of 30 seconds. It is acknowledged and then moved to the cache.

4. With cache plus memory mode, Agent A2 receives a notification and loads the instance of concept C1 in its Rete network. With cache only mode, Agent A2 has to explicitly load the concept when it will be needed for an RTC.

Note that the event E1 is in the cache, but Agent A2 does not load the event in its Rete network. However, if the node (JVM) containing Agent A1 fails, then Agent A2 moves the pending events to its Rete network.

Agent A2 receives an instance of Event E2:

1. Rule R2 executes because agent A2 is aware of the instance of C1. (With cache only mode the instance of C1 is in the Rete network only if it has been explicitly loaded.)

2. In the post-RTC phase, Agent A2 sends out event E3.

Use of Locks to Ensure Data Integrity Within and Across Agents

Objects are managed in a concurrent configuration so that multiple agents can read from and write to the same cache cluster and at times operate on the same set of objects.

Multiple threads in one agent can also behave in a similar manner, to enable concurrent RTCs.

Locking is one of the necessary costs of tuning inference agents for higher performance when concurrency features are used.

Locking is used to ensure the data you read is up-to-date, and to ensure that no other RTC is updating the same data concurrently.

Locking in TIBCO BusinessEvents

The goal of locking is to ensure consistency across concurrent RTCs.

If one RTC has a rule condition that includes a concept, and another RTC updates that concept, then the results could be undefined. Or if two RTCs update the same object at the same time, then different properties of the object could be set by different threads leaving an overall object with incorrect property values.

Locking protects the thread of executing when multiple threads in an agent can cause conflicts by trying to write to the same concept at the same time during concurrent RTCs. The same type of issue can occur across inference agents operating concurrently.

Locking is also necessary to ensure that stale data – data that has been modified in another RTC but not yet written to cache – is not read.

Lock operations do not operate a lock on the object you want to protect itself. They set a common flag that represents a lock — it is up to the developer to ensure that all accesses and updates to a concept subject to locking are enforced, and that only necessary concepts (including concepts that are written to as well as those used in conditions) are locked

The typical lock operation is: in the event preprocessor set the lock, using any unique string as a key. For example, you can use the object extId as the lock string. If a preprocessor cannot acquire the lock (because another event's preprocessor has acquired it) then it waits until either the lock is released OR some timeout occurs.

If an exception occurs after the lock is acquired in the preprocessor and before the RTC completes, the lock is released automatically.

Locking code needs to be prepared carefully. If two events try lock(A) then lock(B) and lock(B) then lock(A) respectively, then a situation can arise where both are waiting on each other's thread. Locking should be used sparingly.

With cache plus memory OM, the need for locks is greater than with cache only, because each agent's Rete network must be synchronized for changes made in all other agents' Rete networks.

Because of issues around effective locking with cache plus memory in multi-agent scenarios, cache only mode is generally recommended. Cache plus memory is useful for objects that change infrequently.

When to Use Locking

Depending on your application, locking may not be required in all cases. However it is almost always needed.

For most applications, use locking in the following cases:

With all modes, for reads.

If you want to read the latest version of a concept in one agent at the same time that another agent might create or update the same concept, mediate the reads through the same global lock that was used when creating or updating the concept. This is done using an event preprocessor and, for Cache Plus Memory, a subscription preprocessor.

With Cache Only mode, for writes.

Global locking is done using an event preprocessor.

With Concurrent RTC (even with In Memory OM or Memory Only mode), for writes.

Multiple RTCs could use the same in-memory object, therefore it needs to be protected using a lock. Use a local lock, not a global lock.

With Cache Plus Memory mode, for writes, and for subscription RTCs.

These RTCs run internally to keep the Rete networks on each agent synchronized with the cache). Locking is done using a subscription preprocessor. It uses the same locking key (string) as in the event preprocessor, if one was used.

With state modeler, for timeouts.

State modeler timeouts do not go through an event preprocessor, so locking is done a different way. This is explained in *TIBCO BusinessEvents Data Modeling Developer's Guide.*

Lock Processing Example Flow

The following example demonstrates common locking requirements.

The example uses these features:

- Cache plus memory mode (requires more locking than cache only mode)
- Cache-aside and backing store (backing store not shown).

Lock Processing
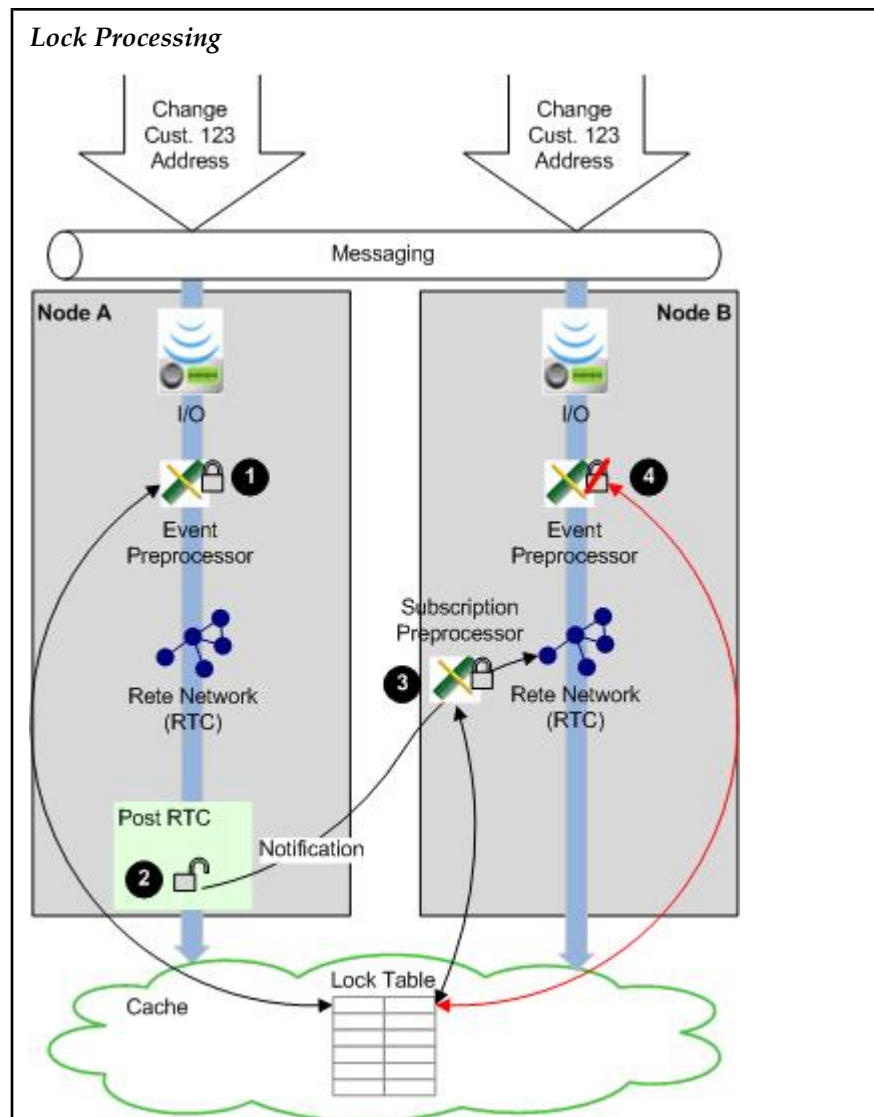
Two agents receive messages that require changes to one Customer instance.

Note that event preprocessors are multi-threaded (see Event Preprocessors for more details).

1. A message comes into a channel on Agent A: a change to a customer address. TIBCO BusinessEvents dequeues the message from the queue, deserializes the message to an event, and calls the event preprocessor function. The preprocessor acquires a lock using the customer's `extID` as the key:

   ```
   Cluster.DataGrid.Lock(Customer@extId, -1, false);
   ```

   This function causes the thread to stop until it gets the lock. In this example, the thread gets the lock.

2. Only one thread handles the RTC. (Concurrent RTC is not used in this example.) Other event preprocessor threads go into a queue. During the RTC, a rule executes and modifies the customer address. After RTC completes, the post RTC phase begins: the address change is written to L1 cache, the cluster cache, and the backing store in parallel. Messages arising from the RTC are sent.

3. After the post RTC actions are completed, the lock is released.

   > 📝 If the write-behind option is used instead of cache-aside, the lock is released after changes are written to cache.

Subscription preprocessor activities run on a different thread from event preprocessor activities. One subscription task handles all the changes to the objects from one regular RTC and executes as

many preprocessors as there are modified objects. (The subscription preprocessor is assigned to an entity in the CDD file Domain Object Override Settings area.)

4. Agent A sends a change notification to all other agents that have subscribed to cluster notifications for this object. Agent B receives the notification, and calls the subscription preprocessor function. It contains the same function shown in step 1 above. It uses the same locking string. Agent B acquires the lock (that is the function returns `true`). The agent updates the Rete network with the changes (using a "subscription RTC"). It will release the lock when the subscription RTC is complete.

5. While the subscription update thread holds the lock, Agent B receives a message with a change to the same customer's address and attempts to acquire a lock, but it is blocked (that is, the function returns `false`).

When the subscription preprocessor releases the lock, then Agent B's event preprocessor can acquire it (depending on timeout behavior) and assert the event to the Rete network.

Depending on timing, either an event preprocessor or a subscription preprocessor could be holding the lock.

Lock and Unlock Functions

TIBCO BusinessEvents provides lock and unlock functions.

Lock Function

The TIBCO BusinessEvents lock function has the following format:

```
Cluster.DataGrid.Lock(String key, long timeout, boolean LocalOnly)
```

If you want to acquire the lock only within the agent, set **LocalOnly** to true. Set the **LocalOnly** parameter to false to acquire a cluster wide lock. For example if you are only concerned about the preprocessor threads in one agent, you can use a local lock. The worker thread that calls the `lock()` function is the only thread that gets blocked.

Unlock Function

All the locks acquired during event processing are released automatically after all post RTC actions, cache operations (and database writes in the case of cache-aside mode) are done.

The format of the unlock function is as follows:

```
Cluster.DataGrid.UnLock(String key, boolean LocalOnly)
```

You can use the corresponding `UnLock()` function for cases where the automatic unlocking does not meet your needs.

The `Cluster.DataGrid.Lock()` and `Cluster.DataGrid.UnLock()` functions are available in event and subscription preprocessors and in rules. However, in general it is not a good idea to use `lock()` in rules as the order of processing of rules is not guaranteed. You can call `Cluster.DataGrid.UnLock()` in a rule only when concurrent RTC is used.

Tips for Locks

The example LockExample (in *BE_HOME*/examples/standard) demonstrates these points, showing use of locks to prevent race conditions.

- Choose an appropriate key for `lock()`. Note that `lock()` does not lock any entity or object as such. The purpose of `lock()` is to ensure sequential processing of related set of objects, but yet ensure concurrent processing of unrelated objects. For example, you want to process messages related to one customer sequentially across the cluster, but want to process messages for different customers in parallel. In this case you could use the customer ID as the key used for `lock()`. This ensures that all messages for a given customer ID are processed sequentially.

- Do not use unchecked and infinite waits (-1) on the lock. The recommended approach is to use the timeout argument, and then exit with an error.

- Always check the return value of `lock()` and if false, either retry or handle it as an error. Don't let application logic proceed if it returns false. Doing so may result in lost updates or stale reads or other such data inconsistencies.

- Try to minimize the locks acquired in one thread. If you have to acquire multiple locks in one thread, ensure that the locks are acquired in the same order of keys, that is, sort the keys.

- Acquire locks before creating instances, to ensure that no other thread creates the same instance.

- Use `lock()` even for read-only operations. If you do not you may get "Inconsistent Database" messages, for example, if there are concurrent deletes elsewhere in other threads or agents.

- In general, avoid using `lock()` in a rule. Since rule order of execution is not guaranteed such usage may lead to deadlocks.

Multiple Keys Protect One Object

In the simplest cases you can use some unique feature of the object you want to protect as the locking key, for example, a customer ID. However different events may point to the same objects using different information. For example, from one channel, the object may be identified using customer ID, and from another, using account ID. In such cases multiple keys are used to identify the same object. When you acquire a lock to protect such an object, you must first get the other key from your system, sort the keys and take a lock on both keys. Sorting can be implemented using a custom function.

If the ordering of keys is not guaranteed, it may lead to a deadlock in a multi-agent or concurrent RTC (multi-threaded) environment. For this reason, avoid use of `lock()` in a loop, where the intention is to process multiple messages. There are other ways to achieve this, for example, using the `assertEvent_Async()` function.

Lock Failures

Instead of throwing an exception after failing to acquire a lock after a few attempts, re-route the event to a special destination that only handles errors (an "error queue"), so you have control over which queue the message goes to.

Write a preprocessor on the "error queue" that does do one of the following for each message:

- Consumes it

- Reports it and then consumes it

- Repairs it and then resends it

For example:
```
System.debugOut("Attempting to lock..");
boolean result = false;
for(int i = 1; i <= 3; i = i + 1){
   result = Cluster.DataGrid.Lock("$lock0", 2000, false);
   if(result == false){
      System.debugOut("Lock acquisition '$lock0' failed. Attempts: " + i);
   }
   else{
      System.debugOut("Lock acquisition '$lock0' succeeded. Attempts: " + i);
      break;
   }
}
if(result == false){
   Event.consumeEvent(newevent);
   Event.routeTo(newevent, "/Channel/LockErrorDestination", null);
   }
}
```
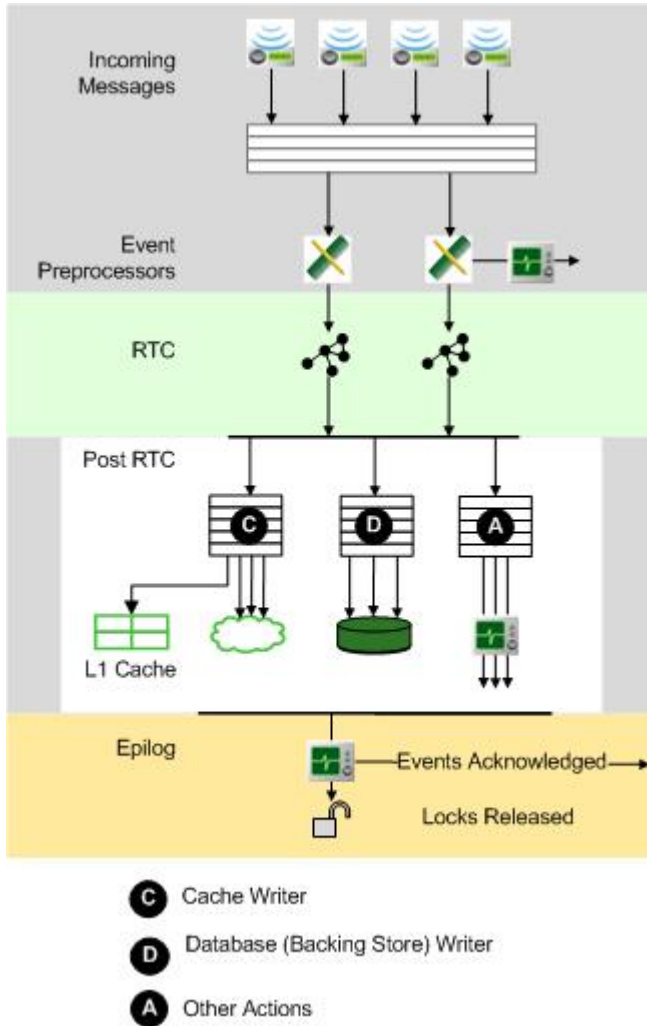
Threading Models and Tuning

Event preprocessing is multithreaded, and for each destination you need to choose a threading option: Shared Queue, Destination Queue, or the Caller's Thread.

Threading Models Quick Reference
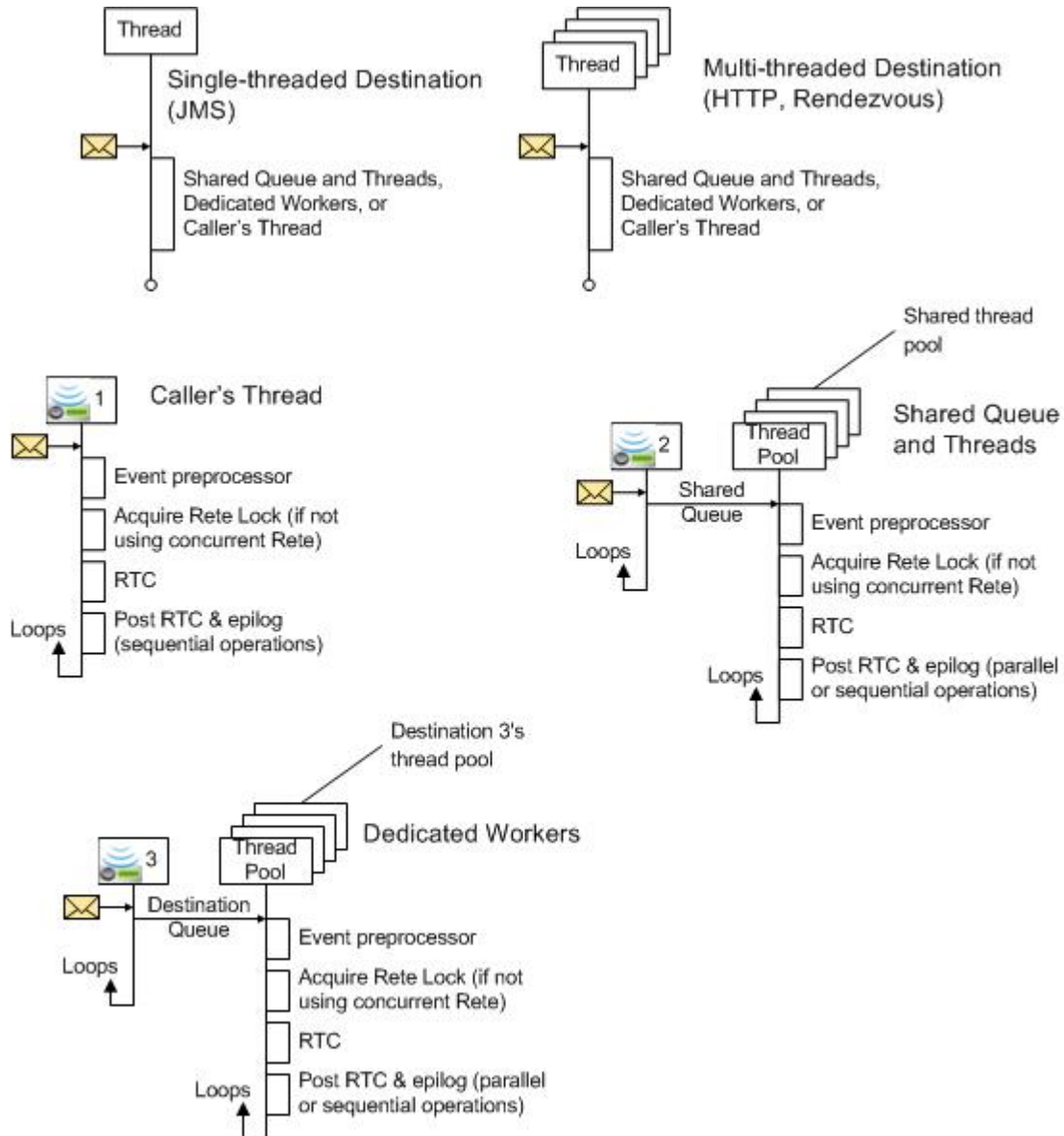
A detailed example for the threading model.

Agent threading example — shared threads, concurrent RTC, cache aside

- Shared Queue and Destination Queue threads are released at the end of the RTC (post-RTC phase uses different threads).

- For the RTC phase, you can choose single or concurrent RTC options.

- For the post-RTC phase, you can choose a cache-aside or write-behind thread management strategy. Cache aside is shown in the diagram above.

- With cache aside you can use parallel operations or (for special cases) serial operations. Use of parallel operations generally requires locking.

- Events that are to be sent out, for example using `Event.sendEvent()`, are actually sent in the event preprocessor or in the post-RTC phase, depending where the function is called.

- Acknowledgements are sent out after the post RTC phase.

  - The exception is events consumed in a preprocessor. In this case acknowledgements are sent immediately.

Scaling Considerations

When you begin to scale up messaging the following are the potential bottlenecks:

- Messages are coming in too fast into the inference engine, or

- Inference engines are not handing off objects fast enough to the cache agents (if write-behind strategy is used) or to the backing store (if cache-aside strategy is used), or

- Cache agents are not accepting the objects fast enough, or

- Backing store is not accepting the objects fast enough.

These points are related. You can add more inference agents and more cache agents to address these issues, depending on where the bottlenecks are occurring.

Event Preprocessor and Rete Worker Thread Options

Event Preprocessor and Rete Worker Thread Options deal with messages that arrive at destinations.

For each destination you need to choose one of the threading model types:

- Shared Queue

- Destination Queue

- Caller's Thread

A thread carries execution to the post RTC phase, at which point execution for the Shared and Destination Queue thread is handed off to another set of threads for writing to cache and backing store. In all threading models, event preprocessing is multi-threaded for high performance.

Diagrams in this section use as an example EMS messages, arriving at JMS destinations.

Each JMS destination creates a separate JMS Session internally, and creates a JMS thread for itself. All queues in all options are blocking, and follow FIFO (first in first out).

Additional tuning may be possible at the event level. For example, the Enterprise Message Service server queue can be sized appropriately. This topic is outside the scope of TIBCO BusinessEvents documentation.

Note that for events sent out in the event preprocessor phase messages are sent immediately, and for events sent out during the RTC messages are sent in the post RTC phase.

Shared Queue

All destinations that use the shared queue threading model share the processing unit's shared queue and threads.

The default and most straightforward option is when one pool of worker threads picks jobs from the shared queue. Execution continues on a thread through to post-RTC.

Shared Queue

The configuration settings are as follows:

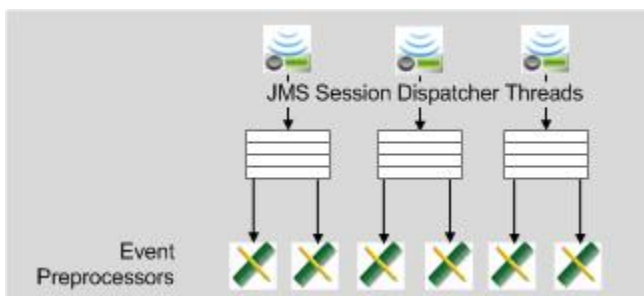| Property	Notes
CDD Editor > **Collections** > **Destinations** > **Threading Model: Shared Queue**	
	Specifies that the shared queue threading model is used.

Property	Notes
CDD Editor > Agents > Queue Size	
	Specifies the size of the queue used for all destinations in the processing unit that use the shared queue threading model.
CDD Editor > Agents > Thread count	
	Specifies the number of threads used for all destinations in the processing unit that use the shared queue threading model.

Advantages	Disadvantages
• Good for multi-core machines, which can make good use of a heavily threaded set-up.	• Too many threads create context switching. • One single destination can become a bottleneck in the case of a sudden increase in incoming messages. • Correlation of events arriving on different queues at different rates can be problematic, as can request-reply situations. • It can be harder to tune performance with only one queue and one set of threads for all destinations.

Destination Queue

The Destination Queue option is similar to the Shared Queue option except that each destination has a dedicated thread pool and set of threads to process messages.

Destination Queue

The following table list the properties:

Property	Notes
CDD Editor > Collections > Destinations > Threading Model: Destination Queue	
	Specifies that the destination queue threading model is used.
CDD Editor > Collections > Destinations > Threading Model: Destination Queue: Thread count	

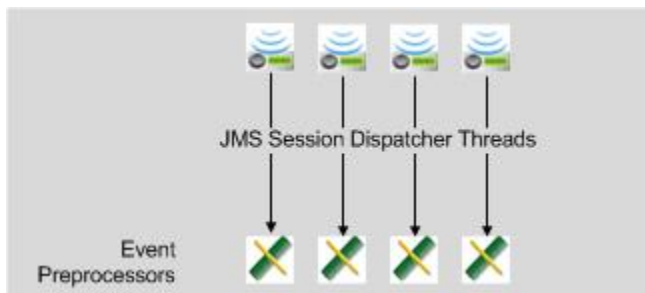| Property	Notes
	Specifies the number of dedicated worker threads for each destination
CDD Editor > Collections > Destinations > Threading Model: Destination Queue: Queue size	
	Specifies the size of the queue used for each destination

Advantages	Disadvantages
• Each destination can be configured differently, to deal with correlation of events arriving at different rates in different destinations, or events that are correlated in different ratios, such as correlation of every tenth event from destination one with every other event from destination two. • If you use priority queues in Enterprise Message Service, you can use dedicated queues to service them efficiently.	• More complex to manage multiple queues and sets of threads.

Caller's Thread

The Caller's Thread option uses the thread (and queue size) provided by the channel resource client, such as the Enterprise Message Service client.

There is one caller's thread per destination. The same thread executes the RTC phase.

Caller's Thread

This option has no tuning controls.

Property	Notes
CDD Editor > Collections > Destinations > Threading Model: Caller's Thread	
	Specifies that the caller threading model is used.

Advantages	Disadvantages
• If the destination uses a single thread (JMS but not HTTP), using this option ensures that the events are processed in the order received. • The messaging library's thread does the message delivery, pre-processing and the Rete operations, so there is less context switching. • The messaging system cannot push events faster than the rate at which it can get consumed, so the system is self-throttling. • Best option for request-reply situations.	• To scale up, many destinations have to be created in order to create that number of caller threads. (And doing so is not always possible, for example, if you need to process messages in the order received.) • Because each destination creates a JMS session, a session might be under use. On some operating systems, sockets and sessions could be very under-used.

RTC Options — Single-Threaded or Concurrent

Depending on your needs, you can choose single threaded or concurrent RTC options, in the CDD Agent settings Concurrent RTC field.
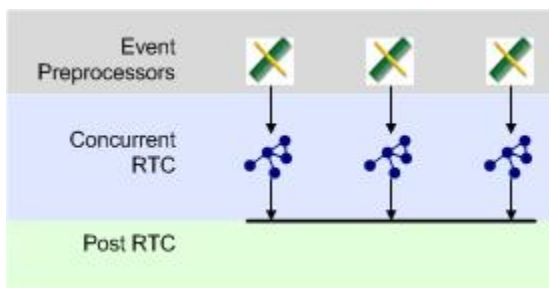
For reference documentation, see *TIBCO BusinessEvents Developer's Guide*.

Concurrent RTC

In the option Concurrent RTC, one RTC executes simultaneously on each thread.

All threads fill post RTC queues. As with any concurrency feature, locking is required.
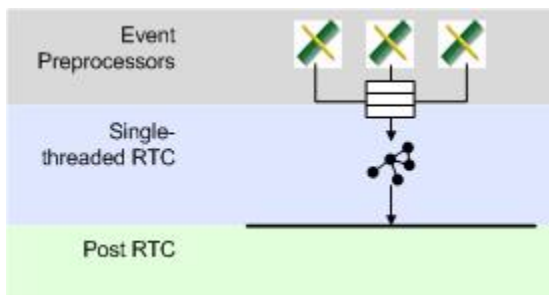
Concurrent RTC

Advantages	Disadvantages
• Can confer performance benefits, given correctly sized hardware and JVM configuration. Best on large high-capacity, high-performance machines. • Does not require cache OM.	• When many smaller CPUs are used, then concurrent agents may give better performance than concurrent RTC. • Requires the same kind of locking as for multi-agent concurrency to protect integrity of the data. The cost of locking negates some of the performance benefits of concurrency. • If you also use concurrent agents, the system can become complex.

Single-Threaded RTC

In the option Single-Threaded RTC, each thread waits to execute its RTC in turn.

Single-Threaded RTC

Advantages	Disadvantages
It's simpler and does not require locking (unless concurrent agents are used).	Lesser performance than concurrent RTC (depending on hardware used).

Post-RTC and Epilog Handling and Tuning Options

Post-RTC and epilog actions are handled by TIBCO BusinessEvents differently in cache-aside and write-behind options.

> For most use cases, use cache-aside strategy, which offers improved controls and is a later implementation than write-behind.

Cache Aside Options

With cache-aside strategy, inference agents manage writes to the cache, to the local L1 cache, and to the backing store in the post-RTC phase.

Cache aside can be used with parallel or sequential operations.

Cache-Aside Options

C Cache Writer

D Database (Backing Store) Writer

A Other Actions
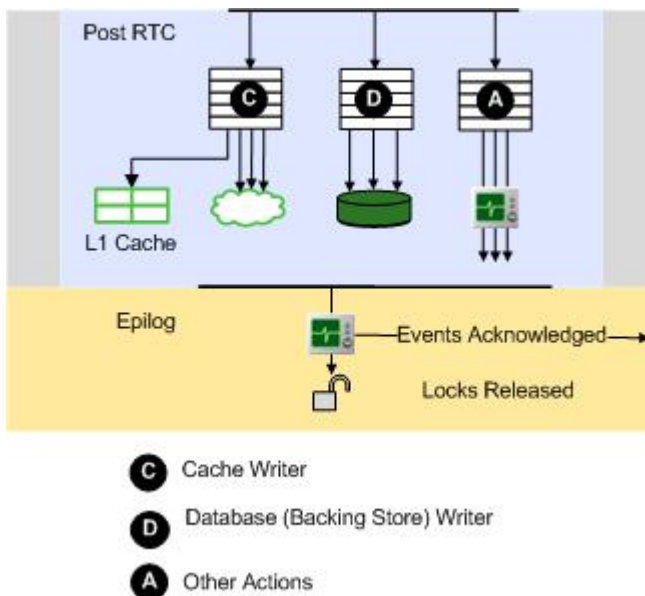
Parallel and Sequential Operations

Use of parallel or sequential operations is set using the following boolean property in the CDD file:

`Agent.`*`agentClassName`*`.enableParallelOps`

Parallel operations is only used with cache aside (defaults to `false` for write behind). It is an agent level property, so you can set it differently on each agent, depending on the needs. For reference documentation, see *TIBCO BusinessEvents Developer's Guide*.

Message Acknowledgment

With both parallel and sequential operations, message acknowledgement and releasing locks (epilog actions) wait for the three post-RTC tasks to complete. Note the following exceptions:

- Events consumed in a preprocessor. In this case acknowledgements are sent immediately.

- JMS messages with acknowledgment types `AUTO_ACKNOWLEDGE` and `DUPS_OK_ACKNOWLEDGE`.

Parallel Operations

When the `enableParallelOps` property is set to `true`, parallel operations are used. Parallel operations is shown in the diagram for this section. It uses multiple threads and queues for more efficient processing during the post-RTC phase.

The parallel operations feature is used only with cache aside. It is enabled by default when both cache aside and concurrent RTC features are enabled.

Sequential Operations

When the `enableParallelOps` property is set to `false`, sequential operations are used. This means that all post-RTC phase operations are done on a single thread. Sequential operations ensures that the system waits to send a reply event confirming that some work has been done, until the result of the work can be seen in the cache.

Advantages

- The following advantages are available when parallel operations is used:

  – Allows batching of RTC database write operations.

  – Provides thread and queue size controls.

- Offers the ability to use the database as the primary storage, and to use cache secondarily, to pass the objects between the Rete network and the database. This strategy is useful in some usage scenarios.

Disadvantages

- There are really no disadvantages in comparison with write behind.

Tuning Properties for Cache-aside Strategy

Tuning properties are set using CDD properties or settings at the appropriate level, depending on the scope.

See *TIBCO BusinessEvents Configuration Guide* for reference details.

Cache Writer Thread Tuning

One cache writer property is available. It is set at the agent class level:

```
Agent.AgentClassName.threadcount
```

Database Writes Tuning

For details about database writer thread and queue tuning see Database Write Tuning Options for Cache Aside.

Parallel or Sequential Operations

For details on the `Agent.agentClassName.enableParallelOps` property, see Parallel and Sequential Operations.

Write Behind Options

With write-behind strategy, cache agents handle writes to cache and to the backing store.

First the inference agent writes data to the cache and then the cache agent or agents write that data to the backing store.

Write Behind Options
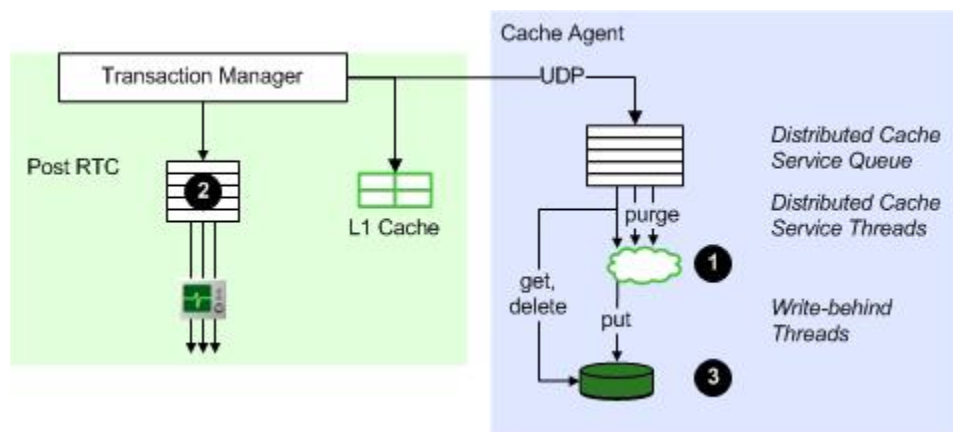
For backing store inserts and updates, one write-behind thread is used for each entity type. Deletes are performed by the distributed cache threads (configurable) and they are synchronously deleted from the database.

Advantages	Disadvantages
• Batches writes during the delay period which increases database call efficiency and minimizes network traffic. • Offloads cache and database work to the cache agent.	• Does not offer database write tuning controls. • Can be slower than cache aside. • If enough cache agents fail, the cache management layer won't be able to persist a write that was done previously, resulting in an inconsistent database.

Tuning Properties for Write-Behind Strategy

Tuning properties for write-behind strategy are used with the Coherence cache provider.

With write-behind strategy, sequential operations are used by default, instead of parallel operations (that is `Agent.agentClassName.enableParallelOps=false`).

There are no tuning properties for write behind and TIBCO BusinessEvents DataGrid. The following properties are used with the Coherence cache provider:

Cache service thread tuning

Uses the following cluster-wide property:

```
tangosol.coherence.distributed.threads
```

Specifies the number of Coherence daemon threads used by the distributed cache service (per processing unit). If zero, all relevant tasks are performed on the service thread. For reference see *TIBCO BusinessEvents Configuration Guide*.

Database writes batch size tuning

When Coherence provides many updates, the backing store commits the writes to the database after each `be.backingstore.commitSize` batch size is reached. For reference see *TIBCO BusinessEvents Configuration Guide*.

Database Write Tuning Options for Cache Aside

Various properties affect how database operations in each RTC transaction are processed and committed to the backing store.

These properties affect the way RTC actions are processed and written to the database when both of these settings are used:

- **Cluster tab** > **backing store settings** > **Cache Aside checked** (true)

- `Agent.agentClassName.enableParallelOps=true`
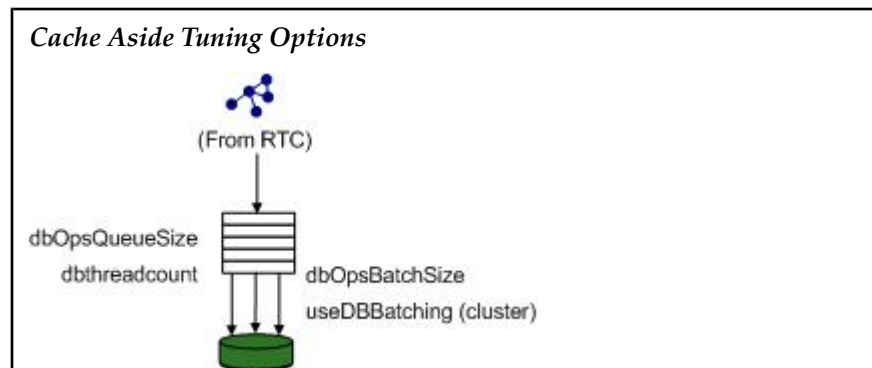
Tuning the parameters in this section can improve performance. However larger values do not imply improved performance. For example, in heavy workload situations, increasing `dbOpsBatchSize` and `dbOpsQueueSize` values results in longer post-RTC times, which in turn delays release of cache locks — which are not released until after the post-RTC

Database Write Queue and Thread Pool (Agent Level)

Actions for one or more RTCs are done in one or more batches, depending on tuning options.

In each batch, TIBCO BusinessEvents does the following actions (as needed):

```
deleteConcepts
deleteEvents
insertConcepts
insertEvents
modifyConcepts
removeObjectTable
saveObjectTable
```

You can tune each agent's database write behavior using the following options:

```
Agent.AgentClassName.dbOpsQueueSize
Agent.AgentClassName.dbOpsBatchSize
Agent.AgentClassName.dbthreadcount
```

The post-RTC database (backing store) transactions are queued into a database-operations queue whose maximum size (set per agent) is defined by dbOpsQueueSize. Each slot in the queue contains all the actions from one RTC.

Database write threads process the RTC transactions from the queue. The number of threads is defined by dbthreadcount.

A database write thread takes up to the dbOpsBatchSize number of RTC transactions, processes them and commits them to the database. (When database write threads are idle, they take available jobs from the database operations queue, even if there are less jobs than dbOpsBatchSize.)

You can monitor JMX parameter AvgDBOpsBatchSize to see the effective value used in your use case.

Database Batching Option (Cluster Level)

All RTC transactions in one batch can be handled as one job (in agents that have the property set to a number larger than one) depending on the cluster-level property setup.

To achieve that, the following cluster-level property has to be set to true:

be.engine.cluster.useDBBatching

and the following agent level property has to be set to greater than one:

Agent.*AgentClassName*.dbOpsBatchSize

In this case, all the RTC transactions in one batch are handled as one job (in agents that have the property set to a number larger than one).

It is important to balance the cost of processing fewer, larger jobs against the gains. The efficiencies gained by setting the useDBBatching property to true are greater if many operations of the same kind apply to the same table. The database writing process has to do an operation for each of update, insert and delete on each database table. So you don't get any gain if only one RTC in the batch is updating, inserting, or deleting in a particular table.

For reference documentation see *TIBCO BusinessEvents Developer's Guide*.

Glossary

A

advisory event

A notice from TIBCO BusinessEvents about activity in the engine, for example, an exception.

agenda

A prioritized list of rule actions that may execute. Also known as the rule agenda. TIBCO BusinessEvents recreates the agenda each time a change in the Rete network requires rules to be re-evaluated, for example, when a simple event is asserted. A rule action in an agenda may disappear without firing when the agenda is recreated, because conditions are no longer met.

agent

TIBCO BusinessEvents operates at runtime using one or usually several agents of different types.

agent class

An agent type, defined in the CDD file, that deploys as an agent instance.

agent group

Multiple deployed instances of one agent class. Used for load balancing and fault tolerance.

assert

Put facts into the Rete network. When object instances or events are asserted into the Rete network, rules may fire as a result.

B

backing store

A disk-based database to provide persistence and a better level of reliability for cache objects.

C

cache

In TIBCO BusinessEvents, a type of object management. Generally refers to distributed storage of facts (objects) in memory.

cache cluster

A group of processing units each running one or more agents configured for use by the cache-based object management type. Some agents are generally configured as cache agents. They store cache data for use by other cluster members.

cache mode

Various cache mode options are available at the object level, for use with cache-based object management. They enable you to tune the performance of your application, and reduce its footprint in memory.

cache agent

A processing unit configured with a non-reasoning agent used as a storage node only. Used with Cache object management.

cache only

One of the cache modes, used at the object level. Instances of entity types that use cache only are serialized and kept in the cache until needed. They must be explicitly loaded into the Rete network when needed for rule processing.

cache plus memory

One of the cache modes, used at the object level. Instances of entity types that use cache plus memory are serialized and kept in cache until needed. Used for objects that change infrequently.

cache server

See cache agent.

channel

A named configuration that allows TIBCO BusinessEvents to listen to a stream of events from a given type of source, for example, JMS messages. A channel contains one or more destinations.

cluster deployment descriptor

XML file containing properties to define the cluster, processing units, and agent classes. Configuration done using TRA file properties in earlier releases is now done using the CDD editor in TIBCO BusinessEvents Studio.

complex event

An abstraction that results from patterns detected among simple events.

Example: A complex event might result from the following simple events that all occurred within one week's time: A stock broker buys shares of xyz stock. The same broker submits a very large order for xyz stock on behalf of a customer. The same broker sells shares of xyz stock at a profit.

complex event processing (CEP)

Correlation of multiple events from an event cloud, with the aim of identifying meaningful events and taking appropriate action.

concept

An abstract entity similar to the object-oriented concept of a class. A concept is a description of a set of properties that, when grouped together, create a meaningful unit. Concepts can be organized in a hierarchical structure.

Example: *Department*, *employee*, *purchase order*, and *inventory item* are all concepts. The term *concept type* refers to the definition and the term *concept instance* refers to the actual object. Concepts are generally created using event data.

concept reference

A property within one concept that references the ID of another concept, known as the referenced concept. A type of relationship between concepts.

conflict resolution cycle

A cycle of activities during which the engine executes one set of rule actions on the currently asserted facts. One RTC may contain multiple conflict resolution cycles.

contained concept

A concept that exists entirely within another concept. A type of relationship between concepts.

custom function

You can add Java-based custom functions as needed to supplement the library of standard functions provided with TIBCO BusinessEvents.

D

decision table

A tabular form presenting a set of conditions and their corresponding actions. A graphical tool for building rules. Used in TIBCO BusinessEvents Decision Manager.

deserializer

A class that performs conversion tasks. In TIBCO BusinessEvents, a deserializer converts messages to events. See also serializer

destination

A channel property that defines a contact point on a given channel. For example, for a TIBCO Rendezvous channel, the destination properties would specify the subjects on which to listen.

distributed cache

In TIBCO BusinessEvents, a form of cache-based object management. In a distributed cache, cached object data is partitioned between the processing units (JVMs) in the cache cluster for efficient use of memory.

E

entity

A concept, simple event, or scorecard. Entity types are the definition of the entity. Similar in meaning to object. The term "instance" generally refers to a concept instance.

event

An object representing some occurrence or point in time.

evict

To remove an object or entry from a cache. An eviction policy defines when an object is removed from the cache.

expires (event)

At the end of the event's time to live period, the event is said to expire. It is removed from the Rete network and (as needed) acknowledged. Other actions depend on the type of object management used.

F

fact

An instance of an event or concept or scorecard in the Rete network.

I

in memory

In TIBCO BusinessEvents, a form of object management. Refers to storage of facts (objects) used by the runtime engine in JVM memory.

memory only (local only)

One of the cache modes, available for cache-based object management configuration. Instances of entity types that use this mode are not stored in cache or backing store and are available only in the processing unit's local JVM memory.

inference agent

In a deployed system, inference agents process incoming events using a Rete network and a set of rules that are triggered by conditions in incoming events. Inference agents in Cache OM systems allow fault tolerance and load balancing.

instance

Similar to the Java term "object instance." By custom, applied only to concepts, though event definitions have object instances also.

L

L1 Cache

Used with Cache OM, the L1 cache is a local cache that gives inference agents quick access to recently used objects.

lambda transition

A transition without a condition. This term is used in state model configuration.

O

object management (OM)

The aspect of TIBCO BusinessEvents that deals with management of all the facts used in the runtime engine. Often shortened to OM in documentation.

ontology function

TIBCO BusinessEvents generates ontology functions for each entity type in a project. There are three types of ontology functions: *constructors*, to create a simple event or concept instance; *time events*, to create and schedule time events, and *rule functions*, to invoke rule functions.

P

payload

Similar to a JMS message, a simple event can contain properties and a payload. The payload holds the content of the message. You can define the XML schema for the payload when you configure the simple event definition. Payloads can also contain strings and Byte arrays.

processing unit

Definition of a TIBCO BusinessEvents engine which runs in one JVM. Contains agents and other properties.

Q

query agent

A query agent is a non-reasoning agent that and has read-only access to the underlying objects in the cache cluster. A query agent has no Rete network. Available only with TIBCO BusinessEvents Event Stream Processing add-on software.

R

Rete algorithm

Dr Charles L. Forgy developed the Rete algorithm for expert systems.

Rete network

An in-memory network of objects based on the Rete algorithm which enables fast matching of facts with rule dependencies.

"The word 'Rete' is Latin for 'net' or 'comb'. The same word is used in modern Italian to mean network. Charles Forgy has reportedly stated that he adopted the term 'Rete' because of its use in anatomy to describe a network of blood vessels and nerve fibers. . . . A Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side (the condition part) of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern." (http://en.wikipedia.org/wiki/Rete_algorithm)

retract

Remove facts from the Rete network.

RMS

Rules Management Server. The server component of TIBCO BusinessEvents Decision Manager add-on software. RMS manages the rules management repository.

Run to completion (RTC) cycle

A run to completion (RTC), cycle generally begins when an external action causes changes to the Rete network. It ends when there are no more rule actions to execute as a result of that initial change (and any subsequent changes caused by rule actions). One RTC is composed of one or more *conflict resolution cycles*. Other terms for RTC are `forward chaining` and `inferencing`.

rule

A declaration, with a set of conditions and actions. If all the conditions in the rule are satisfied by facts in the Rete network (and the rule is at the top of the agenda), TIBCO BusinessEvents executes the action.

rule based time event

See time event.

rule function

Custom functions written using the TIBCO BusinessEvents rule language and using a provided user interface.

Also used to refer to a type of ontology function.

rule session

An older term that has been replaced by the term inference agent.

S

serializer

A class that performs conversion tasks. In TIBCO BusinessEvents, a serializer converts events to messages.

simple event

An object representing a business activity that happened at a single point in time. A simple event includes information for evaluation by rules, metadata that provides context, and a separate payload — a set of data relevant to the activity.

simple event definition

A description of the channel, destination, properties, and payload for a simple event.

standard function

A library of standard functions is provided with TIBCO BusinessEvents for use in rules and rule functions.

T

time event

A type of event definition, used as a timer. Two types are available: rule based, and interval based.

time to live

A simple event property that defines the delay after which a simple event expires.

U

UML (Unified Modeling Language)

A language that asTIBCO BusinessEvents Architect's Guidesists in building a diagram of any complex entity. TIBCO BusinessEvents diagrams use the UML. The TIBCO BusinessEvents term, *concept*, is similar to a UML class.

V

virtual rule function

A rule function whose signature is defined in a TIBCO BusinessEvents project and whose implementation is defined using decision tables. For use by TIBCO BusinessEvents Decision Manager add-on software.

W

working memory

The runtime processing area for rules, objects, and actions. Rules apply only to data in the working memory. Often used to mean Rete network.