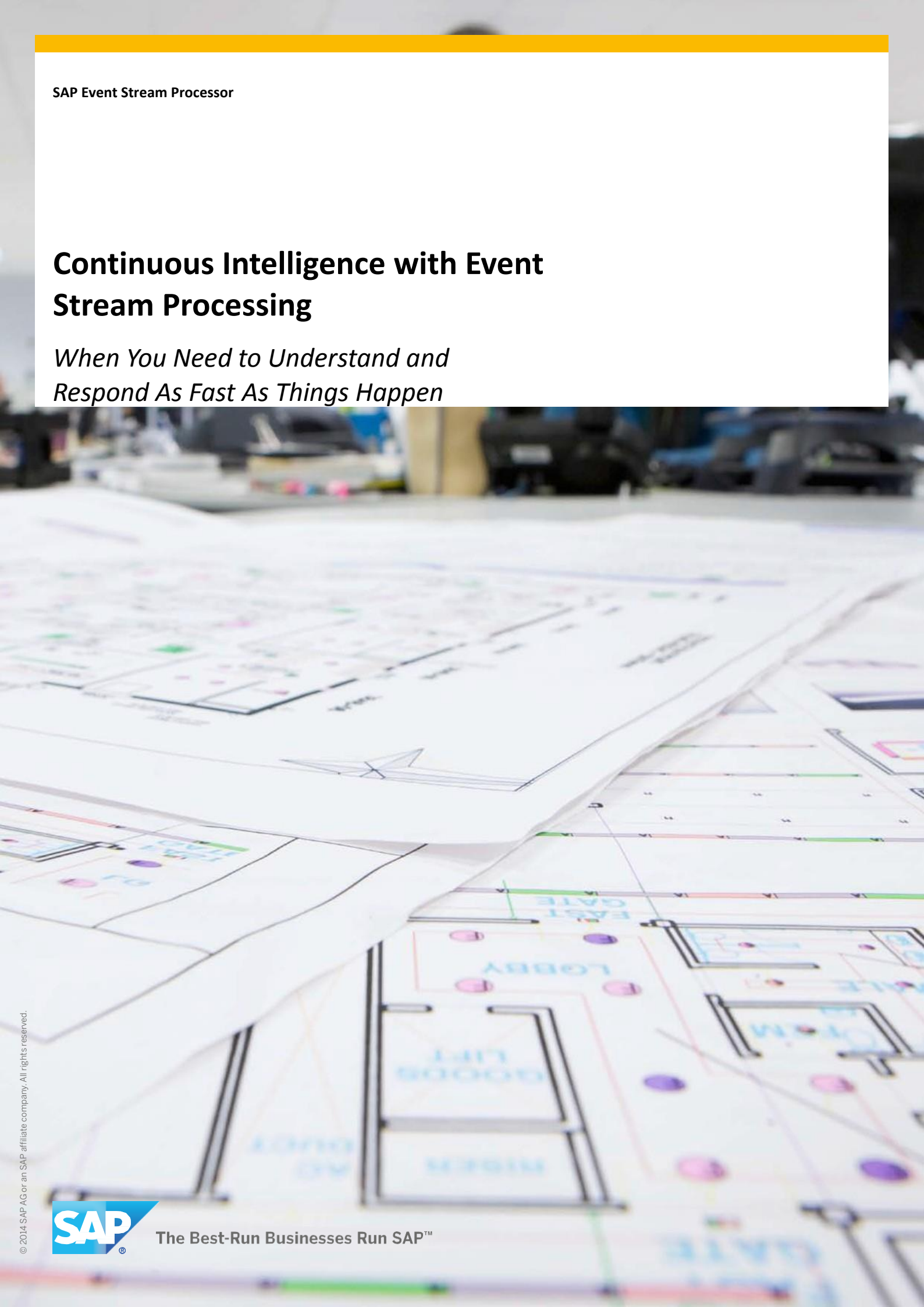
SAP Event Stream Processor

Continuous Intelligence with Event Stream Processing

When You Need to Understand and Respond As Fast As Things Happen



The Best-Run Businesses Run SAP™

Table of Contents

Introduction to Event Stream Processing

SAP® Event Stream Processor (SAP ESP) is a high-performance event processing engine designed to make it easier for organizations to implement continuous intelligence solutions. Part of the SAP data management portfolio, it enables rapid application development and deployment, with a scalable event processing engine for immediate analysis of fast-moving data. This paper explores the applications of event processing for real-time insight and describes SAP ESP technology and architecture.

INSIGHT FROM DATA AS IT ARRIVES

Businesses today benefit from - and are also burdened by - the wealth of data at their disposal. Systems that automate and facilitate every aspect of the business operation produce, collect, and send information – at ever-increasing volumes and rates. Smart devices are proliferating, constantly sending information about conditions and actions. In the capital markets, market price data is the lifeblood of trading, and it arrives at the rate of hundreds of thousands of messages per second. A web site produces a steady stream of "clicks" as the customers interact with the site. And IT systems of all types are logging events for everything that happens.

Making sense of all this data can be daunting. There's the inherent complexity of combining a variety of data from a diverse set of sources and then analyzing the raw data to extract actionable insight. The traditional approach to this is to collect all the data in a database or data warehouse and then use business intelligence (BI) tools to analyze it.

Increasingly, however, there is business value in being able to respond faster to new information. Velocity has value. Rather than analyzing historical data to understand what happened last week or even yesterday, what if you could understand what is happening right now? Even better, what if your systems could tell you when something is happening that deserves your immediate attention? What if your systems could automatically respond to the new information, so you don't have to?

Applications that continuously absorb, analyze, and respond to streams of information as fast as things happen are delivering *continuous intelligence*, enabling a "right now" response across a range of industries.

COMPLEX EVENT PROCESSING

Consider the following situations:

- Sensor data from equipment is monitored for trends or correlations that indicate a problem, alerting an operator to take immediate action before equipment damage occurs
- A commodity pricing application continuously adjusts quoted prices in response to market conditions – where delays mean either lost business or lost profit.
- IT system events are continuously monitored to watch for patterns that indicate a possible security threat
- User actions on a web site are analyzed to determine the best offers to show, not just based on historical data for the user but also considering current context

These are just a few examples of the types of applications that can benefit from event stream processing, a type of complex event processing (CEP). The common denominator of these applications is that they share the need to continuously collect, process, and analyze data in real time, producing results without delay, even when the data arrives at very high rates. CEP technology analyzes incoming events in the context of other events as well as other available information, turning raw data into useful insight.

CEP technology extracts insight from incoming events as fast as the information arrives, producing results in real-time – within milliseconds of the arrival of new information.

Instead of analyzing historical data to understand what happened last week or yesterday, what if you could understand what is happening right now? Even better, what if your systems could tell you when something occurs that deserves your attention?

Traditional databases were designed to process individual transactions at very high rates. Analyzing data to look for specific conditions, or deriving higher-level summary data, were tasks that had to be done offline, using query tools that were never designed to produce actionable intelligence in real time. SAP HANA makes on-demand analysis of large data sets possible, but how do you collect all that data? And what if you want to be alerted to trends, or spot patterns in real time? Performing analysis on static data means that insight is delivered when someone asks a question – as opposed to when something happens. In many cases this is too late – the window to act has already closed.

Event stream processing technology delivers the data analysis tools traditionally provided by relational databases or even spreadsheets, but in a real-time, event-driven implementation that can process incoming data at very high rates and produce results with near-zero latency.

Think of it as taking some of the fundamental concepts of a relational database and turning them upside down. A traditional relational database is designed to collect data and store it, where you can then use analysis to filter the data, combine it, group it, search for patterns, derive high-level summary data, and so on. The analysis takes place offline and not in response to incoming events. An event processor, in contrast, takes incoming messages and streams them through a set of predefined, continuous queries to produce derived streams or sets of data.

We call them "continuous queries" because the data analysis logic is similar to what might be included in a traditional database query.

Examples of queries are:

- "Show me which events meet this criteria."
- "Tell me if this pattern of events (or nonevents) occurs."
- "Show me the current total of all events matching these criteria."
- "Group events by these values and calculate the average for each group."

While continuous query logic may be similar to a traditional database query, the implementation is anything but. Event processing uses a dataflow architecture to pass incoming messages through the continuous query operators as soon as the message arrives, so that the result sets are instantly updated. The functions used within the continuous queries have been implemented in a way to maximize throughput and minimize latency.

An Alternative to Custom Code

SAP ESP provides an alternative approach to building high-performance enterprise-class applications that must process event data in real time. Custom applications written in C++ or Java are expensive and time-consuming to build. They are also typically brittle and therefore expensive to maintain, since the processing logic is hard-coded and tightly bound to the data structures. What's more, designing and writing highly efficient code for real-time processing requires specialized programming skills.

Using SAP ESP, it is possible to build and deploy new applications in as little as 20% of the time that it would have taken using Java. That's an 80% reduction in cost, and it means getting the new application into production in one-fifth of the time. And that's only the initial benefit. Once the application is in production, there is far less code to maintain, meaning less ongoing support costs.

The need for Agility

A by-product of building applications on SAP ESP is greater agility. Because the business logic is implemented using high-level authoring tools and is separated from the underlying data-handling infrastructure, it can be rapidly changed as the needs of the business change. Organizations using applications built with ESP can literally change the processing logic in minutes. And because ESP is based on an event-driven architecture, an ESP application is decoupled from the inputs and outputs, reducing dependencies across the various systems that produce and consume data.

A Spectrum of Uses and Requirements

SAP ESP is used to address a variety of use cases, wherever there is fast moving data and value from processing it in real-time. But regardless of the use case, they all have in common the need to perform one or more of the following functions.

Situation Detection

Incoming events are monitored to detect trends or patterns that indicate the existence of an immediate opportunity or an imminent problem. This can range from a simple filter to a complex set of rules that aggregate or correlate incoming events. When the situation is detected, a high-level (complex) event is instantly published – with information about the situation. This can be sent as an alert to users or can be used to initiate an immediate response.

Continuous Computation

Incoming events are correlated, grouped, and aggregated - and computations are then applied to produce new information such as summary data, high-level statistics, or adjustments to key operating parameters. This data can be displayed on live operational dashboards, can be available to applications for immediate on-demand access, or can be sent to applications or systems to adjust current operations.

Examples of this type of CEP include:

- Continuously compute and update indices or other operating parameters
- Continuously update key performance indicators (KPIs)
- Continuously update valuations or exposures
- Continuously aggregate data across multiple sources to be able to see the "big picture"

Data Collection – Event Persistence

SAP ESP is often used to capture high speed event data in SAP HANA or other databases. Beyond simply capturing raw data, business logic can be applied to the incoming data to filter, transform or enrich the data, capturing useful data in the most meaningful form. ESP can also be used to help manage the size of the data set being captured. For example: high value data can be captured in SAP HANA, while the rest is diverted to Hadoop. ESP can also apply rules to correlate events, capturing compound rather than individual events, or can sample the data.

Application Integration with Intelligent Event handling

Many applications are built on an event-driven architecture, but the basic tools for this provide the mechanisms for the exchange of event data without providing the ability to analyze that data. ESP can provide intelligence within an event-driven architecture to analyze events in the context of other events as well as other current information. In this way, it can determine what new events need to be generated or which action should be taken based on an event.

In any particular context, you may find the focus to be on one particular aspect of event processing. Yet recognizing that different applications have different requirements will help ensure that you select the optimal tool or tools for the job.

Design Patterns for Complex Event Processing

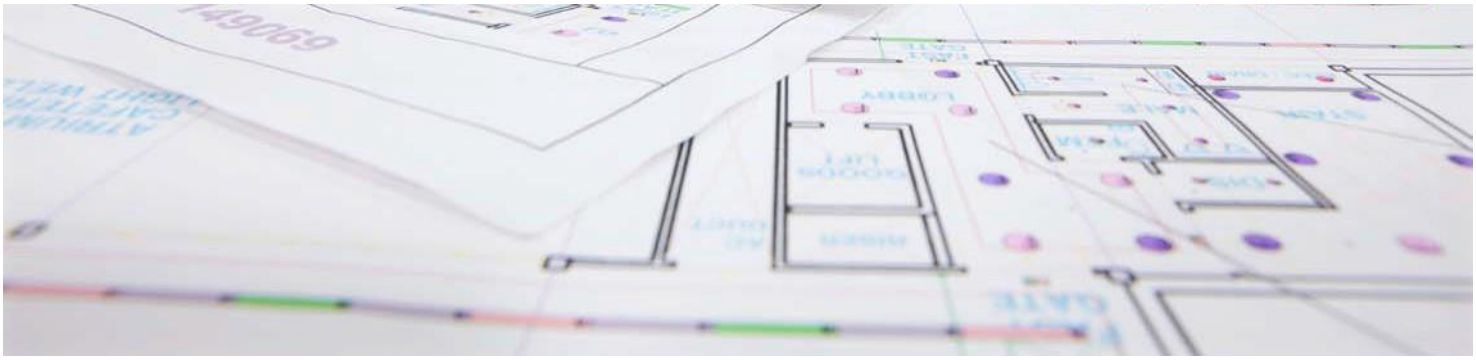
When we talk about the ability to analyze incoming event data in real time, we are actually referring to a variety of functions that can be applied to the data, alone or in combination, to derive high-level intelligence or to trigger a response. You can:

- Apply simple or complex filters to data to detect conditions of interest. This can include correlation of events across multiple sources, correlation of events across time, and watching for sets of events that match a defined pattern.
- Combine data from multiple sources, including the ability to combine streaming and static data or to combine data that arrives at different times.
- Define data retention "windows," based on either time or number of elements, across which the computations will be performed.
- Group and aggregate data, producing high-level summary data and statistics. This can include trends (moving averages), net positions or exposures, and so forth.
- Compute new data elements: enrich simple event data by adding new fields that are computed based on context, data from other sources, or other recent events.
- Watch for specific patterns of incoming events, generating a new event when the pattern is detected. Patterns can even include the absence of events. The new event can be used to trigger a response or to generate an alert.
- Transform data format and structure. This can go beyond simple message-level transformation to create entirely new events based on individual or multiple events using rules that take into account context, reference data, and so on.
- Generate high-level events from patterns or groupings of low-level events.

Respond as fast as things happen

SAP ESP extends the power of SAP HANA with event-driven action. As a real-time platform, SAP HANA enables on-demand analysis of data that is updated in real-time. With SAP ESP, action can be taken as soon as things happen, based on insight from the incoming event streams. And SAP ESP can be used to continuously update the information in SAP HANA, ensuring that users and applications are always operating on the basis of current information.

SAP ESP is designed for speed and scalability, so that it can process and analyze data as fast as it arrives – even if the data is arriving at the rate of hundreds of thousands of events per second. And regardless of the incoming data volume, every event is evaluated as soon as it arrives and results are published within milliseconds of the arrival of the new information.



EXAMPLES OF SOLUTIONS USING EVENT STREAM PROCESSING

Organizations use event processing technology in a wide range of different applications across a number of industries. Here's just a sample of some of the types of applications that are leveraging event stream processing for real-time insight and response.

Internet of Things (IoT)

- Actively monitor data arriving from sensors and smart devices
- Generate alerts when immediate attention is warranted
- Predictive maintenance: alert operations staff to imminent failure
- Situational marketing: target offers to users based on context
- Consolidate live status information in operational dashboards
- Collect high-value data in SAP HANA for use by application as well as historical analysis and reporting

Manufacturing – collect and monitor machine data

- Predictive maintenance
- Process optimization
- Operational monitoring and alerting

Energy

- Short-term demand forecasting
- Smart-meter monitoring
- Demand management and response
- Oil and gas production monitoring

Telecommunications

- Real-time system and network monitoring – Traffic volumes and problem detection
- Fraud detection – Spotting unusual traffic patterns in real time
- Customer retention – Proactive customer service
- Marketing – targeting offers to context

Financial Services

- Trading and risk: real-time valuation and exposure monitoring
- Market data aggregation
- Trade monitoring
- Real-time index calculation
- Fraud detection and prevention

Retail - eCommerce

- click-stream analysis for real-time offer optimization
- fraud detection
- customer experience management

Public Safety

- Crisis management – alerting and updating operational dashboards with data from the field
- Infrastructure monitoring – early warnings by analyzing sensor data in real-time

Applications that continuously absorb, analyze, and respond to streams of information as fast as things happen are delivering continuous intelligence, enabling "right now" response across a range of industries.

SAP Event Stream Processor

SAP Event Stream Processor is a high-performance, enterprise-class CEP engine that can be used to quickly implement and deploy a wide range of applications that need to analyze and act on event data in real time. It is fifth-generation CEP technology and represents the state of the art in complex event processing. The software combines performance, versatility, and ease of use to meet the requirements of the most demanding environments. SAP ESP is part of the SAP data management portfolio, providing a comprehensive infrastructure for cutting-edge applications that enable real- time business like never before.

KEY CONCEPTS

SAP ESP is designed to process and analyze streams of events. So that's a natural starting point.

In the real world, an event is something that happens. In an event-stream-processing context, an event is actually a message that contains information about a real-world event. An event could be an order arriving in a database, a price update on a stock market ticker, a user clicking on a Web link, or a sensor reporting a piece of information.

Each **event** (message) has a schema – a set of fields with a defined data type for each field. In database terminology, an event is a "tuple." In fact, because many ESP concepts are adopted from databases, we refer to the fields in an event message as **columns**.

An event stream can be thought of as a channel that delivers events that have a common structure or schema. In ESP terminology, we generally refer to these simply as streams. Thus, streams of event data are a key starting point for any ESP project. Event streams flow into an ESP server into either an input stream or an input window.

Streams in an ESP project are stateless; that is, a single event enters, passes through the stream, and is "forgotten" once it is passed downstream. But you are limited in the types of analysis you can do on a single event. Therefore ESP lets you create windows.

Windows in an ESP project have state. They are very similar to a table in a database, except that they are constantly changing. Incoming events add a row to the window, update an existing row in the window, or delete a row from the window.

The columns in the window match the schema of the incoming events. The size of the window can be set in three ways:

- number of rows – A window set to a fixed number of rows will shed older rows when new rows are added once the size limit is reached.
- Time – A window with a time policy will keep rows for the defined time period and then delete them. So a one-minute window will keep each row for one minute after it arrives (or is last updated) and then delete it.
- Self-managed – Since events don't just add rows to a window but can also modify or delete existing rows, a window can be self-managed. No size or time limit is set, but rather incoming events will delete existing records in the window. An example of this would be an order book that has a set of open orders. New orders are added to the book, and when an order is filled or canceled, an incoming "cancel" or "fill" event removes it from the book.

All windows have **primary keys**. A primary key consists of one or more fields in the row that uniquely identifies that row in the table.

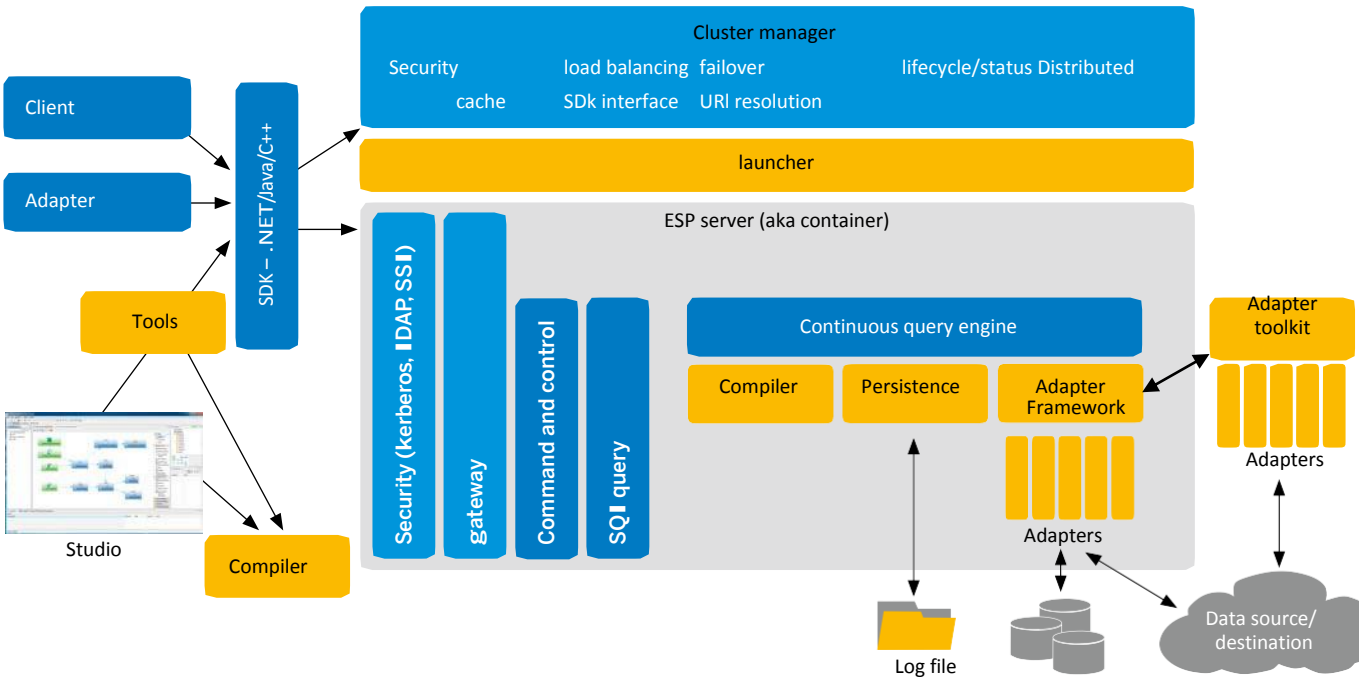
Events can (optionally) have explicit operators, or "opCodes." This is a unique aspect of SAP ESP, though it is not at all unique in the broader scope of data processing. Incoming events may directly update information maintained in a window, and they can contain an opCode that indicates how they should be applied to the window. Thus, an event with an opCode of "insert" adds a row to the window, an "update" event updates an existing row with the same key, and a "delete" event deletes the row with the indicated key value.

Continuous queries take input from one or more streams or windows and apply an operation to produce a new stream or window. Streams and windows that are produced as the result of a continuous query can be output streams or windows or they can be local streams or windows, performing an intermediate step on the overall processing of the event.

An ESP **project** is what runs on an ESP server. It defines a set of input streams or windows and a set of continuous queries that produce one or more output streams or windows. Projects are written in **CCL** – the event processing language used by ESP. CCL is derived from SQL, making it familiar and simple to use.

An ESP application is a complete set of components that addresses a business need. It will consist of one or more projects and one or more adapters, and it will often include other components such as a user interface, a dashboard, a database, and so forth.

figure 1: SAP® Event Stream Processor Architecture overview



ARCHITECTURE OVERVIEW

SAP Event Stream Processor is a platform for building applications that need to process and analyze streams of event data in real time. It consists of a number of components (as presented in Figure 1 and described below).

- ESP projects run on an ESP server – or "cluster". This can be a single machine or a multi-machine cluster. The cluster is managed by the ESP cluster manager that receives and manages requests to start and stop projects. Each project runs in its own container – an independent server process that is started by the cluster manager. The cluster manager is also responsible for restarting projects after a failure of the project or the machine it's running on. Any number of cluster managers can run – all are peers, which avoid a single point of failure.
- The ESP studio provides the design time tools used to build and test ESP projects. It is based on Eclipse and is available as both a stand-alone application as well as a plugin for the SAP HANA Studio. As a design-time component, the studio does not need to be running in production systems. While a project can be deployed from the Studio, it can also be deployed onto a production cluster directly from the command line, the ESP Cockpit or from scripts.

  In fact, even for creating ESP projects, the studio is entirely optional; a project can be written in CCL using a text editor and then compiled and deployed using command-line utilities.
- The ESP compiler takes one or more CCL files to produce an executable project in the form of a "ccx" file. The ccx file is what gets deployed to the server. The compiler can be invoked from the studio or the command line.
- The ESP publish/subscribe (pub/sub) interface is used to stream data into an ESP project or to subscribe to data being published by an ESP project. It can also be used to load static data into an ESP project. It is part of the software development kit (SDK) for SAP ESP, which is available for C/C++, Java, and .NET.
- ESP adapters are typically used to connect ESP projects to common sources and destinations. ESP comes with a wide range of pre-built adapters as well as an adapter toolkit to easily build custom adapters.

- The SQL query interface allows snap-shot SQL queries against ESP windows via the ESP ODBC driver. This also supports SAP HANA smart data access – allowing ESP windows to be included in HANA queries.

The ESP Cluster

ESP projects run on an ESP cluster. You can think of an ESP cluster as an ESP "cloud" that can encompass any number of physical servers (see Figure 2). When you run an ESP project, you run it on the cluster, rather than on a specific machine, and the cluster manager assigns the project to a physical machine.

In its simplest form, an ESP cluster can consist of a single ESP cluster manager running on a single machine. When you instruct the cluster manager to run a new project, it will be started on the same machine. But as additional machines are added to the cluster, the cluster manager will make use of all available machines when starting new ESP projects.

Each ESP project runs as an individual process on an ESP cluster. When starting the project, the cluster manager starts an instance of the ESP server running the desired project.

In a multi-machine ESP cluster, a project is started on the ESP cluster, not on a specific machine. The cluster manager assigns it to a machine. Also, there will normally be two or more ESP cluster managers, running on different machines, to avoid a single point of failure. Cluster managers run as peers, so the loss of one or more cluster managers does not affect the operation of the cluster as long as there is at least one cluster manager running.

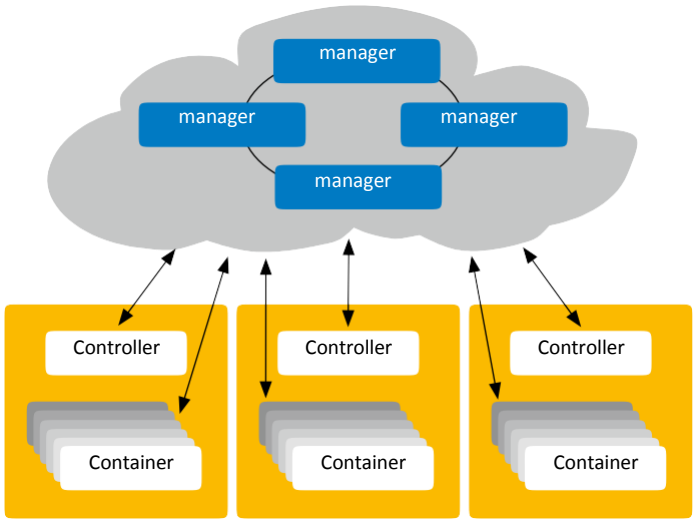
Machines in the cluster that do not have a manager will have an ESP controller that acts as an agent to manage local projects under the auspices of the cluster managers.

Some features of an ESP cluster include the following:

- A single cluster can be limited to a single machine or can span any number of machines.
- An ESP cluster can run any number of projects simultaneously, subject to the capacity of the available hardware resources.
- Connections to streams or windows in projects are made via a URL consisting of the server.workspace.project.element. The cluster manager resolves the URL to a specific host and port.
- Projects run in workspaces, providing namespace protection.
- Projects can be started and stopped independently of all other projects running on the server.
- Projects can be designated for automatic failover or active- active configurations (see the "Resiliency" section).
- The cluster manager is fully redundant – Multiple cluster managers will run as peers, avoiding any single point of failure.
- Clusters can span multiple locations – The machines in a cluster do not have to be co-located. Intra-cluster communication is all based on Transmission Control Protocol (TCP).

Resiliency

At the level of the ESP cluster, as long as there is more than one machine in a cluster, there is no single point of failure. While a cluster can run as a single node on a single machine, where resiliency is desired there should be at least two machines in the cluster, each with a cluster manager.



A by-product of building applications on a CEP platform is greater agility. Business logic is employed with high-level authoring tools and separate from the data-handling infrastructure, so it can be rapidly changed as needs of the business change.

ESP Project Resiliency

There are three options for resiliency when running an ESP project. The desired option is specified in the project configuration file (or can be specified at runtime).

- no resiliency – If the project fails, it must be manually restarted.
- Automatic failover – If the project fails, the cluster manager will restart it. Projects send heartbeats to the cluster managers to allow detection of a "hung" project.
- Active-active – Projects configured for active-active will run with redundant live instances, one as a primary and one as a secondary. Normally, the primary and secondary will run on separate machines. All connections from outside the cluster are directed at the primary while it is alive. Data between the primary and secondary is continuously synchronized. If the primary fails, all outside connections will automatically be redirected to the secondary.

Security

SAP ESP has built-in security features for authentication, access control, and encryption. User authentication is done at the cluster level, and there are three types of authentication that are supported:

- Lightweight Directory Access Protocol (LDAP)
- RSA
- Kerberos

SAP ESP supports secure socket layer (SSL) encryption of all communication between the ESP cluster and client applications (including both data sources and consumers).

Role-based access control can restrict what individual users or groups can do. Authorization can be used to restrict activities at both the cluster level and the project level; within a project, it can be used to restrict access to individual streams and windows. Thus, based on the users' authenticated identity, access control can restrict whether they can:

- Administer the cluster
- Start or stop projects
- Write to or subscribe to a particular stream or window in a named project

INTEGRATION

One of the powerful aspects of SAP ESP is the decoupled nature of an event-driven architecture. Integration into an existing environment is typically nonintrusive. It can go in as an overlay to existing systems without needing to replace or reengineer those systems.

Integration with SAP ESP is via input and output adapters. SAP ESP comes with a set of common adapters that are part of the base product. SAP offers additional specialized adapters as add-ons, and custom adapters can be readily built using the ESP Adapter Toolkit or the SDK.

Each adapter normally runs as stand-alone process, in which case the communication between the adapter and the ESP server is via a TCP socket. External adapters can run on the same machine as the ESP project or on a different machine.

Standard Adapters

The SAP ESP engine ships with the most common adapters included. These simply need to be configured to connect to the relevant source or destination. We are constantly adding new adapters, so please see the online list for adapters included in the product. These adapters include:

- message buses – Connect to a message bus to receive in- coming events or publish events. Supported message buses include Java Message Service (JMS), TIBCO, and IBM Web- Sphere MQ.
- Databases – Load data from or publish data to databases via ODBC or JDBC. Most major databases are supported. Input data can be loaded at start-up, or the database can be periodically polled.
- Web Services: SOAP, REST and WebSockets
- file – Read input events from a file; write output events to a file, with CSV and XML parsing, one time or polled.
- Socket – Receive events on a socket or write events to a socket, with CSV and XML parsing.

Specialized Adapters

A number of specialized adapters are available from SAP as add-ons and are licensed separately. These include:

- Financial market data
- Financial Information eXchange (fIX) protocol

Custom Adapters

A custom adapter can easily be developed using the ESP Adapter Toolkit. The toolkit provides a Java framework that supports pluggable transport and parse/format modules. The toolkit ships with a collection of pre-built modules and custom transports or parsers can be written in as little as a few hours.

In addition, adapters can be written using the ESP SDK directly. This allows adapters to be written in C, C++, or .NET, in addition to Java

A custom adapter can also be developed without using an ESP API by simply writing events to or receiving events from a socket or file in an ESP-supported format.

Software Developers Kit

The SAP ESP SDK is available for Java, C, C++, and .NET. It supports both publish and subscribe capabilities to allow custom adapters to be built. What's more, while we generally talk about "adapters," in fact the SDK can be embedded directly in an application that publishes to or subscribes to SAP ESP, or both. What's more, most of the capabilities of the SDK are also available via the ESP REST interface.

The SDK is designed to provide a simple way of:

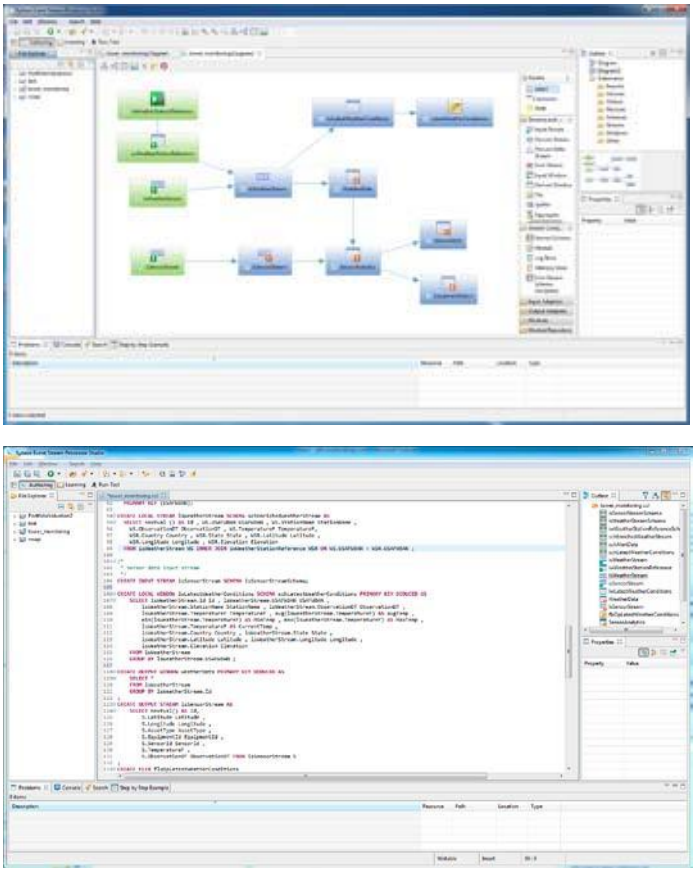
- Connecting to an ESP cluster
- Constructing an event with a schema corresponding to the stream or window that will receive it
- Publishing an event to a specific stream or window in a specific project
- Subscribing to an output stream or window in a live project
- Parsing the event received from an output stream or window

THE ESP STUDIO – MAXIMIZING PRODUCTIVITY

The ESP studio is built on the Eclipse framework, providing a familiar environment to many users. It can run as a stand-alone application or as a plug-in to the SAP HANA Studio. It supports capabilities to build, test, tune, debug, and run ESP projects. It is designed to be:

- Easy to learn, even for nonprogrammers –Projects can be created and run by technically proficient users who aren't programmers. The studio is designed to be accessible to these users as well as experienced programmers.
- Efficient to use – After all, SAP ESP is a productivity tool. Therefore, it's important that the studio is designed with user productivity in mind. The goal is for experienced users to feel that it's an efficient environment in which to work.

figure 3: visual and CCL editors



Two editors: visual and text

The studio in SAP ESP takes an innovative and unique approach to authoring by providing a choice of two different styles of editing in a smoothly integrated environment (see Figure 3). The visual editor allows the user to build a project by selecting building blocks from a palette, arranging them on a diagram, adding connectors to direct data flow between, and configuring the behavior of each operator. The CCL editor is a more traditional programming environment – a syntax-aware text editor. The unique aspect of the studio, however, is that the user can flip between the two editors at any time; changes made in one are reflected in the other.

Designed for speed and scalability, SAP ESP can process and analyze data as fast as it arrives – even at the rate of hundreds of thousands of events per second. And it can deliver results within milliseconds of the arrival of new information.

Many users will choose to build their projects primarily in the visual editor, but they may find it useful to occasionally look at or even directly edit the underlying CCL. Likewise, some users will find it more efficient to build projects directly in CCL, but they will occasionally find it useful to switch to the visual editor to see the data flows within the project. This is especially true for large or complex projects where the visual editor can serve as a useful navigation tool within the project.

CONNECTING TO DATA SOURCES

Most projects begin with one or more data sources. Normally, at least one data source will be a true "stream" of events as they happen. There may be more than one streaming source, and there may be other sources that provide either static or semi-static data (that is, data that updates infrequently).

The ESP server contains a number of built-in adapters that can be used to connect to common data sources. We also provide a number of optional add-on adapters, and custom adapters can be built using the adapter toolkit or the SDK.

Within the studio, a user can select an adapter from the palette, add an adapter to the project, and configure the adapter. The adapter will then be controlled by the ESP server. Note that the adapter integration framework allows custom adapters to appear in the palette alongside the standard adapters.

To simplify the process of building new projects that will use data from existing sources, the studio includes a feature called schema discovery that lets the user easily select a schema from an existing source, import it into the project, and, if desired, create an input stream or window connected to the source and with the schema for that source. Some adapters also provide features to allow for custom field mappings between the source and the input stream.

The Run-Test Perspective

The ESP studio is a complete integrated development environment (IDE) with tools for both authoring and testing. The run and test perspective includes tools to start and stop projects, monitor and view live projects, and test, debug, and tune projects.

Specific tools include:

- **Server view** – Connects to any number of ESP servers, enabling users to see running projects and start and stop projects
- **Stream viewer** – Allows users to see the output from any stream or window, updated in real time, in a tabular view
- **Performance monitor** – Provides a graphical view of the project with color coding to indicate throughput rates or queuing; updates dynamically, providing ability to identify bottlenecks
- **Event tracer** – Lets the user view how an event affects each query node in a project
- **Debugger** – Enables setting breakpoints in a project
- **Record and playback** – Records data coming into input streams of a project, capturing it in a file that can then be played back any number of times. Playback speed can be the rate at which the data was originally received, or it can be slower, faster, as fast as possible, or at a fixed rate.
- **File upload** – Allows data to be loaded into input streams directly from data files
- **Manual input** – Allows users to manually input individual events, setting the value in each field

CCL: THE CONTINUOUS COMPUTATION LANGUAGE

CCL is the primary event processing language of SAP Event Stream Processor. ESP projects are defined in CCL, which defines the inputs and a set of continuous queries to produce the desired outputs.

CCL is an adaptation of SQL with extensions for event streams. It provides the same type of sophisticated data analysis functionality you find in SQL, including the ability to filter, group, aggregate, and join data across streams. However, CCL also includes features that are required to manipulate data during real-time continuous processing, such as defining windows on data streams, watching for patterns of events, and invoking custom event handlers.

What really sets CCL apart from SQL is that CCL is designed for expressing continuous queries. A normal SQL query against a relational database executes once each time it is submitted to a database server and must be resubmitted every time a user or an application needs to re-execute the query. By contrast, a

CCL query is continuous. Once it is defined in the project, it is registered for continuous execution and stays active indefinitely. When the project is running on the ESP server, a registered query executes each time data arrives from one of its data sources.

Although CCL borrows SQL syntax to define continuous queries, the ESP server does not use an SQL query engine. Instead, it compiles CCL into a highly efficient byte code that is used by the ESP server to construct the continuous queries within the data-flow architecture.

CCL Example

Figure 4 shows a simple example of how CCL is used to define two continuous queries.

Common CCL Query Types

A CCL project consists of one or more input streams or windows and one or more derived streams or windows. Each derived stream or window includes one or more continuous queries that take data from inputs and produce the derived result.

Most continuous queries do some combination of the following operations:

- Filter – Filter an input stream, only passing through the events that pass the filter criteria.
- Aggregate – Group incoming events according to a common "key" value, producing a single output record for each group. One or more fields in the output record will typically be computed using aggregate functions such as sum(), count(), average(), or even more advanced statistical functions such as standard deviation or correlation.
- Compute – Transform an incoming event into an event with a new set of fields where the new fields are computing from the fields of the incoming event, possibly along with other information.
- Join – Join events or records from two or more inputs based on common field values. You can join a stream to a window, join two windows, and even do multi-way joins. This can be used to enrich incoming events with additional reference information, or it can combine data from multiple event streams, matching the events within a defined time window.
- Pattern Detection – Watch for a specific pattern of events, and generate a new event when the pattern is detected. Patterns are defined with a time interval and a sequence or combination of events, and they can include missing events – for example, a pattern might be "eventA followed by eventB, but NOT eventC – within 10 minutes."

Figure 4: Query Definition with CCL

Window of last 10 minutes of price updates

```
CREATE InPUT wInDow PriceFeed kEEP 10 min
SChEmA (Id integer, Symbol string, Price money,
Shares integer, TradeTime date)
PRImARy kEy (Id);
```

Apply inserts, updates, deletes to maintain current positions

```
CREATE InPUT wInDow Positions SChEmA
(BookID string, Symbol string, SharesHeld
integer)
PRImARy kEy (BookID, Symbol);
```

Compute moving average

```
CREATE loCAl wInDow VWAP
PRImARy kEy DEDUCED AS
SEIECT
    PriceFeed.SymbolSymbol,
    PriceFeed.Price LastPrice,
    (sum(PriceFeed.Price * PriceFeed.Shares)/sum
    (PriceFeed.Shares)) VWAP,
    PriceFeed.TradeTimeLastTime
fRom PriceFeed
gRoUP By PriceFeed.Symbol;
```

Join the positions to the prices

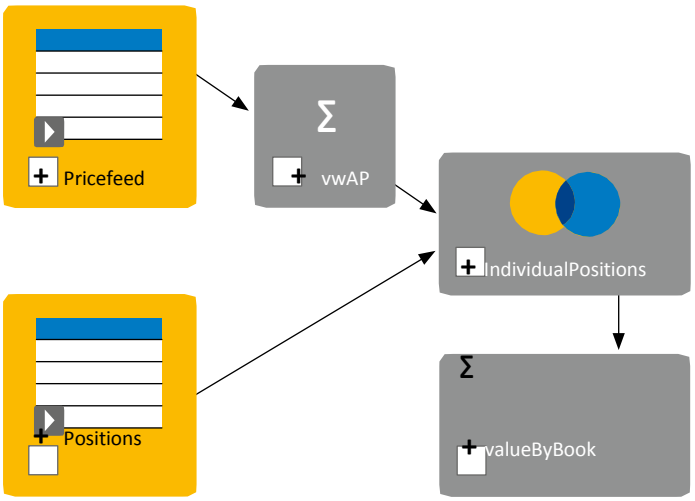
```
CREATE oUTPUT wInDow individualPositions
PRImARy kEy (BookID, Symbol)
AS SEIECT
    Positions.BookID BookID, Positions.Symbol
    Symbol, (VWAP.LastPrice *
    Positions.SharesHeld) CurrentPosition,
    (VWAP. VWAP * Positions.SharesHeld)
    AveragePosition
fRom Positions, VWAP
whERE Positions.Symbol = VWAP.Symbol;
```

The CCL Query graph – Data-flow Programming

An ESP project can consist of a single input stream followed by a single continuous query, but a project will often contain multiple continuous queries. Large projects may contain dozens of inputs and hundreds of continuous queries.

ESP uses data-flow programming to direct incoming data into and through a set of continuous queries. Thus, a project is broken down into a sequence of continuous queries, each of which further refines the incoming data into the set of desired outputs. It's very easy to represent this visually (using, for example, the visual editor) as a set of streams and windows, with the data flowing between them. Thus, a typical query graph of an ESP project might look like Figure 5.

figure 5: Typical Query graph in an Event Stream Processor Project



SPLASH Scripting for Extensibility

SQL is a great starting point for the most common functions used in processing event streams. It has the benefits of familiarity and the simplicity of a declarative language. But there are times when the event processing logic you need to implement can't be easily expressed in an SQL query. That's where SPLASH comes in. It brings extensibility to CCL.

SPLASH is a simple procedural scripting language that has a syntax similar to Java, but its spirit is closer to languages such as AWK or Perl that solve relatively small programming problems. It provides procedural control, the ability to iterate using WHILE and FOR loops, and data structures for holding data from one event to the next.

SPLASH makes it unnecessary to go outside SAP ESP to use Java or C++ to write custom functions or operations. It has the benefit of being:

- Easy to learn – Because SPLASH is a simple language, there is a small learning curve. You don't need to be a Java programmer to use it.
- Designed to maximize user productivity – SPLASH is embedded in-line in CCL files, so there is no need to leave the ESP studio or use separate testing tools.
- Efficient – SPLASH is designed for high performance in processing incoming events, leaving the user to focus on the business logic.
- Safe – SPLASH exposes only the tools needed to process events, thereby limiting opportunities for error.

SPLASH Controls and Data Structures

SPLASH includes controls such as FOR, WHILE, and IF, giving the user complete procedural control with the ability to iterate over data sets. SPLASH includes the ability to define variables that retain state from one event to the next. It also includes advanced data structures that are designed for efficient handling of events and event sets, such as:

- **Event Cache** – An alternate windowing mechanism for working with sets of events that allows greater control than the standard CCL KEEP clause
- **Dictionaries** – Sets of key or value pairs for efficient lookup
- **Vectors** – Ordered sets of values or data structures

Flex operators

Flex operators are custom, programmable event handlers that go beyond what you can do with the standard relational operators of CCL. They are used to create a new derived stream or window in the same way that a continuous query does. However, instead of passing the incoming event through a SELECT statement to produce the result, a flex operator invokes an event handler written in SPLASH upon the arrival of a new event. Thus, a user can write

CCL is an adaptation of SQL with extensions for event streams. It provides the same type of sophisticated data analysis functionality found in SQL, but it also has features that are required to manipulate data during real-time continuous processing.

sophisticated custom operators in SPLASH and not limit processing to the set of operations that can be achieved via the standard SQL relational operators.

Built-In function library

Each continuous query will include a set of "column expressions" to compute the value of each field in the new event being produced. These expressions can draw from the extensive set of built-in functions:

- Math
- Statistical analysis
- Geometry
- Logical
- String
- Date and time
- Calendar
- Aggregation
- Sets
- Bitwise
- XML

User-Defined Functions in C, C++, or Java

External function libraries in C, C++, or Java can be used in ESP expressions once the library has been loaded. The interface specification is defined in the CCL Programmers Guide for SAP ESP.

Modularity

An ESP project can invoke external CCL modules. This facilitates reuse and team development, where different team members can be responsible for different modules. It can also be used to improve the maintainability of large complex projects.

A project can use a CCL module defined in another CCL file by first using the IMPORT command to include the CREATE MODULE statement from the external file. It then uses the LOAD MODULE statement to invoke the module, and a single module can be invoked more than once within a project. When a module is loaded, the input and output bindings are set. Additionally, any parameters used by the module are set when the module is loaded. Thus, a module that computes a moving average could have the duration set via parameter such that it could be loaded one place to produce a 10-minute moving average and loaded another place to produce a two-hour moving average.

DATABASE INTEGRATION

The output from SAP ESP is often captured in an external database, such as SAP HANA or SAP IQ. What's more, it's often necessary to use information stored in external databases to process an event. For example, a continuous query that is processing an incoming event that contains a customer ID may need more information about that customer. This additional reference information might come from a customer database that is indexed by customer ID.

Event data – either raw events or derived events – can be captured in an external database by attaching one of the ESP database output adapters to the CCL streams/windows with the data to be persisted.

To bring data from a database into the ESP project, you can use one of the following methods:

- Preload data from the database using the ESP database input adapter. The data can be loaded into an ESP Window or other data structures when the project starts, and the adapter can be optionally configured to refresh the data at a regular interval.
- As needed: data from an external database can be fetched when it is needed to process a new event. With SAP HANA, this can easily be done by defining a CCL REFERENCE, which is a proxy for a table or view in HANA. This allows streams to be joined to HANA tables/views. Alternatively, the database can be queried from CCL using the getdata() function, which is available for all supported databases. When an incoming event triggers execution of the expression containing the getdata() function, an SQL query is made against an external data source, the data is retrieved, and it can then be used to compute the expression.
- Streaming – The SAP Replication Server can be used to turn transactions in external databases into event streams in real time, streaming them into the ESP server via the ESP replication server adapter.

DESIGNED FOR PERFORMANCE – HIGH THROUGHPUT WITH LOW LATENCY

SAP ESP was designed from the ground up for speed and scalability, and to be capable of meeting the performance needs of the most demanding applications. On a two-CPU Linux server, for example, an ESP server can process well over 100,000 messages per second. Additional hardware can easily scale to process well over 1 million events per second on a single server. Latency, measured from the time a message arrives until processing is complete and results have

been produced, is typically in the range of a fraction of a millisecond to a few milliseconds. To minimize latency, SAP ESP uses real-time data-flow architecture that moves the data through continuous queries. Unlike databases that store and then query, the entire ESP architecture is event driven and designed to flow data through the system as quickly as possible. All data is held in memory, and even where windows need to be recoverable, a proprietary high-speed log store mechanism is used to provide data recoverability with minimal impact on speed.

The ESP engine has also been designed to deliver consistent latency. When latency is measured, some systems may display low average numbers but with a wide range of values, including spikes that can run into seconds. For applications where latency is critical, SAP ESP can be counted on to deliver a consistent latency profile.

For scalability, SAP ESP leverages a 64-bit multithreaded architecture to fully leverage the parallel processing capabilities of multicore and multi-CPU servers. To further extend this, the ESP cluster architecture manages any number of ESP projects running across all available servers with dynamic data bindings to efficiently pipe information between different projects. Projects can scale beyond the boundaries of a single machine by splitting into multiple interconnected subprojects, each running on a different server in the cluster.

DESIGNED FOR VERSATILITY

As described earlier, different applications have different needs. Many CEP products are designed to address a single type of application. For example, there are a number of CEP "rules engines" that are designed expressly for situation detection. That's fine if all you need is situation detection, but the technology may not be extensible to other types of applications. SAP ESP was designed to address the widest possible range of event processing requirements. It can:

- Monitor incoming data streams for conditions that represent opportunities or threats
- Augment data streams with data from other sources or computed values
- Group data by different dimensions, producing high-level summary data or statistics
- Consolidate data from multiple heterogeneous systems, forming a single aggregate view or stream
- Operate on large data sets spanning large time windows
- Collect raw or results data for use in historical analysis or reporting, or to provide an audit trail

Some of the specific aspects of the ESP architecture that give it this versatility include:

- native state management – Incoming messages can be processed as inserts, updates, deletes, or "upserts." This lets SAP ESP efficiently manage data windows where in- coming messages don't just represent a new data point in a time series but instead represent an update to previous in- formation. Many CEP implementations don't handle updates and deletes – they treat all incoming messages as new data points in a time series. The reality is that many data streams produce updates, changes, and cancellations. Whether it's a change to an order book or a correction to data previously sent, these updates need to be applied to previously received data to maintain an accurate view of the current state.
- SQL with SPLASH for extensibility – CCL, derived from SQL, provides familiarity and ease of use. The in-line SPLASH scripting language provides extensibility, making it easier to add custom operators and functions with the fine-grained control of a procedural scripting language with data structures for efficiently maintaining state between events.
- on-demand queries – ODBC driver execute snapshot queries against ESP windows. Includes support for SAP HANA smart data access.

SAP ESP runs on a clustering architecture that allows any number of projects to be run on a single ESP "cloud" that can span any number of machines.

Many CEP products are designed to address a single type of application. SAP ESP was designed to address the widest possible range of event processing requirements.

- Rich subscriptions – Each subscription to an ESP output stream can be tailored to the needs of the consuming application. Subscriptions to windows can deliver the full window followed by updates, or just updates. Subscriptions can include an SQL SELECT statement to further filter, aggregate, or otherwise refine the data.
- Security – Built-in security including access control, authentication, and encryption
- Database integration – SAP ESP has strong integration with databases both to capture event data and to load context from databases for use in processing events. And SAP replication technology can be used for real-time change capture on databases, turning database transaction into real-time event streams.

Part of the SAP Data Management Portfolio

HIGH-PERFORMANCE, EXTENSIBLE, SCALABLE APPLICATIONS

SAP ESP is part of the SAP data management portfolio, an integrated set of data management tools with SAP HANA at its core. The SAP Platform comprises a revolutionary, integrated platform for the next generation of real-time applications. Optimized for both transaction processing and real-time analytics with unprecedented scale, the platform reduces total cost of ownership by combining functionality that was previously unavailable in a single integrated platform.

Event Stream Processing for SAP HANA

SAP ESP can be deployed with SAP HANA for high- speed capture of streaming data as well as enabling immediate response to changing conditions. The software can capture data in SAP HANA at the rate of over one million messages per second. SAP ESP can also apply "smart capture" rules to filter, cleanse, and enrich data before storing it in SAP HANA.

SAP ESP employs several techniques to enable data capture at such high rates, including:

- Microbatching for optimal SAP HANA load performance
- Multithreaded parallel loading against partitioned tables
- Use of insert arrays

Beyond data capture

At the same time that SAP ESP collects data, it also analyzes incoming data to generate alerts, initiate immediate responses, or stream new insights to downstream applications.

For example, SAP HANA can supply data to SAP ESP to enrich incoming events with reference data or to compare current trends against historical norms. What's more, SAP ESP can tap the analytics functionality of SAP HANA. To help assess the significance of new information, the engine can access the SAP Predictive Analytics software library.

SAP IQ for data warehousing

ESP supports SAP IQ for long-term persistence of event data at extreme scale. SAP IQ efficiently manages vast amounts of data – petabytes and beyond – with unique patented compression functionality that reduces the size of raw input data by 50% to 70%. With its advanced compression, organizations can afford to capture and store all event data, or they can use SAP ESP to perform "smart capture" – only capturing meaningful events.

SAP's vision is to provide a single, logical, real-time data platform that provides the framework for business transformation at a pace that suits your organization – and delivers innovation without disruption.

www.sap.com/contactsap

The Best-Run Businesses Run SAP™

CmP22162 (14/09)