A System for Monitoring Mobile Networks using Performance Management Events

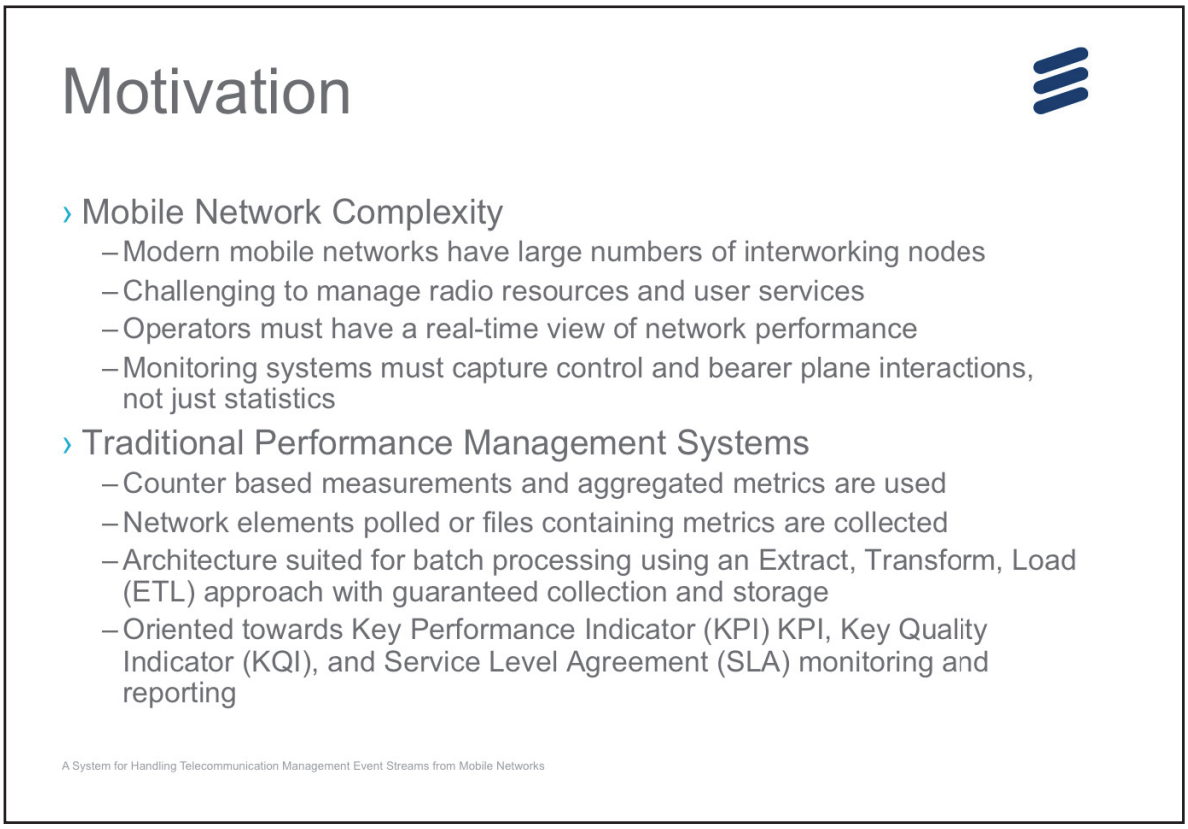
Sajeevan Achuthan and Jimmy O'Meara
Ericsson Ireland

The predominance of mobile broadband data traffic has driven a significant increase in the complexity and scale of mobile networks. The orthodox performance management approach is for network elements to periodically report counters and metrics to performance management systems that use batch processing architectures to produce delayed reports on KPIs, KQIs, and SLAs [1] based on those counters and metrics. Mobile network operators have recognised the shortcomings of this approach and are demanding systems that provide a much more granular view of networks in real time [2]. Performance measurement events from networks [3] and user equipment [4] provide information at the required level of granularity but management systems must be evolved to use those events to monitor networks in real time.

This paper presents a performance monitoring system that uses network events to monitor mobile network performance in real time. The system has an architecture that distributes the incoming event load efficiently by filtering events at their ingress point, distributing only events that are used in the system, and using Complex Event Processing to correlate and aggregate events. The system uses a model driven approach to so that all components in the system have a common understanding of the events being processed and produced in the system.

The experiences gained in deploying a proof of concept implementation of the system in three live networks are outlined. Real time network monitoring proved to be very effective in those deployments, and the system performed efficiently in real network conditions.

978-3-901882-50-0

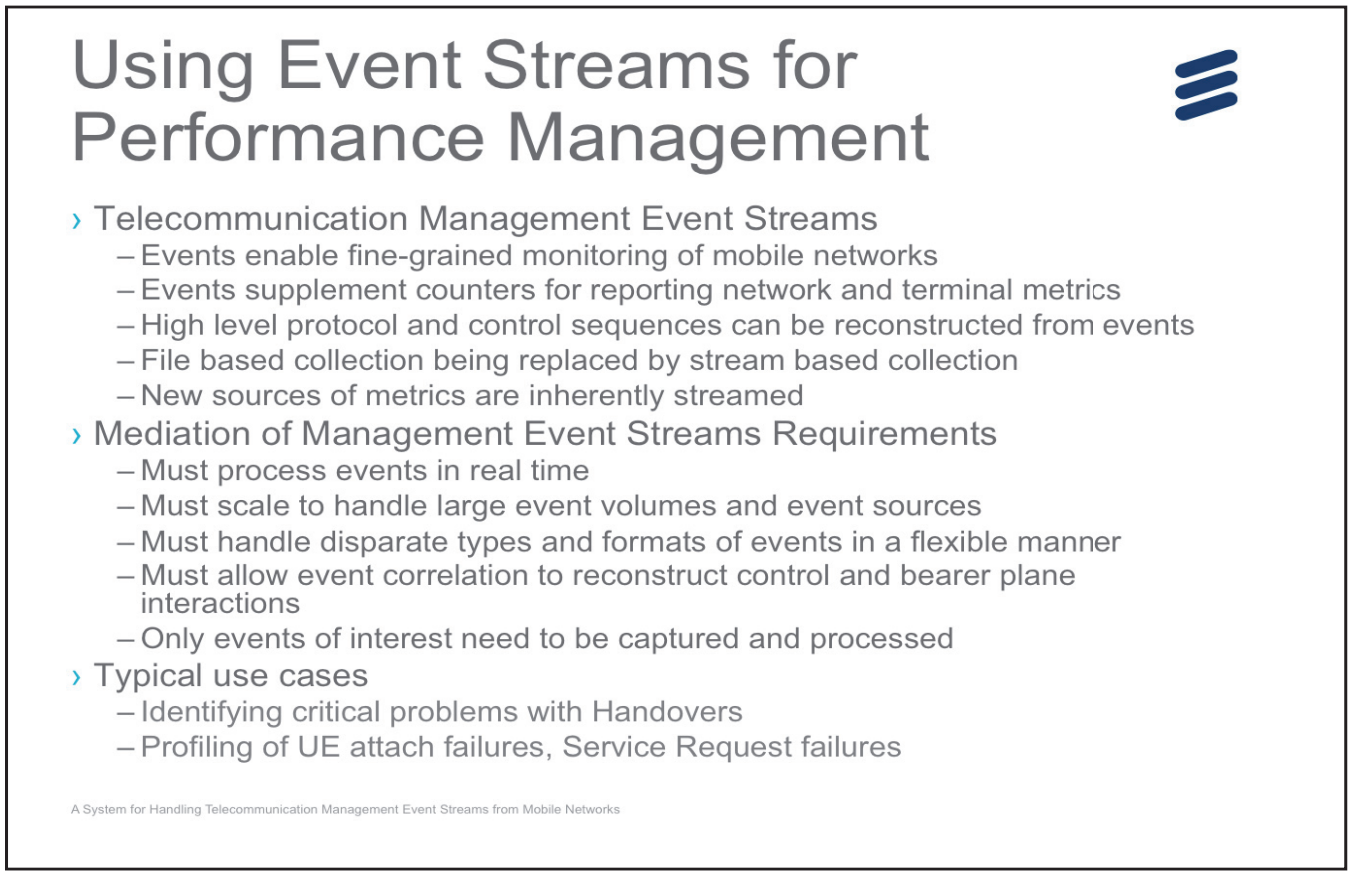


In the last few years, mobile broadband data has replaced voice as the predominant traffic type in mobile networks [5]. Mobile data traffic is doubling annually, a trend expected to continue for some time [5][6]. Mobile networks have evolved with this change, with LTE networks designed primarily to carry data traffic rather than voice traffic [7]. The number of base stations in modern mobile networks is much larger than in first and second generation networks because higher cell densities increase the amount of traffic that can be carried. Problems with complex signalling interactions used for procedures such as inter-domain handovers and user bearer management are difficult to identify and troubleshoot. Network operators have recognized the challenges that managing such networks pose [2], and have driven the adoption of Self Organising Network (SON) features such as Automatic Neighbour Relationship (ANR) management [8]. This combination of large traffic volumes, large network sizes, complex procedures, and SON-based self configuration mean that mobile networks are highly dynamic and complex; it is imperative that operators can monitor networks in real time to observe interactions as they happen, to identify and troubleshoot problems, and to plan optimizations.

The 3GPP specification LTE network metrics [9] is typical of a counter approach to performance measurement; aggregated metrics summarise network performance. Counters report on the number of attempts and successful attempts of procedures such as handoffs, and average and maximum bitrates are reported for throughputs on channels. Counters are an efficient approach to network monitoring; network elements calculate counters from raw information, with just counter information being stored and reported. Counters for a reporting period (ROP) collected periodically from network elements [10]. Two disadvantages of counters is that raw information on network performance is not available for analysis and there is an inherent delay of at least one ROP before counter data can be collected and processed.

Performance management systems typically implement the Service Quality Management process of eTOM [11]. They are data warehousing, analysis, and reporting systems that use file based metric collection mechanisms. Their primary purpose is to collect and store metrics with a high degree of reliability and to track and report on KPIs, KQIs, and SLAs[1].

The main contribution of this work is that, for the first time, it applies event streaming, complex event processing, and model driven development together to monitor and analyse telecommunication event streams, thus allowing real time performance management of mobile networks. Events from both Radio Access Networks and Core Networks can be analysed together to provide full correllated analysis of mobile network performance.



Event-based metric collection, where network elements report metrics on significant events in bearer and control sessions, are increasingly being used to provide more fine grained information than is available using counter based metrics [3] and can be used to supplement information received in counters or as a standalone source of information if the management infrastructure can support the volume of events generated. An event is recorded at each significant point in a procedure or control sequence, so their performance can be examined in detail and individual sequences can be reconstructed for troubleshooting.

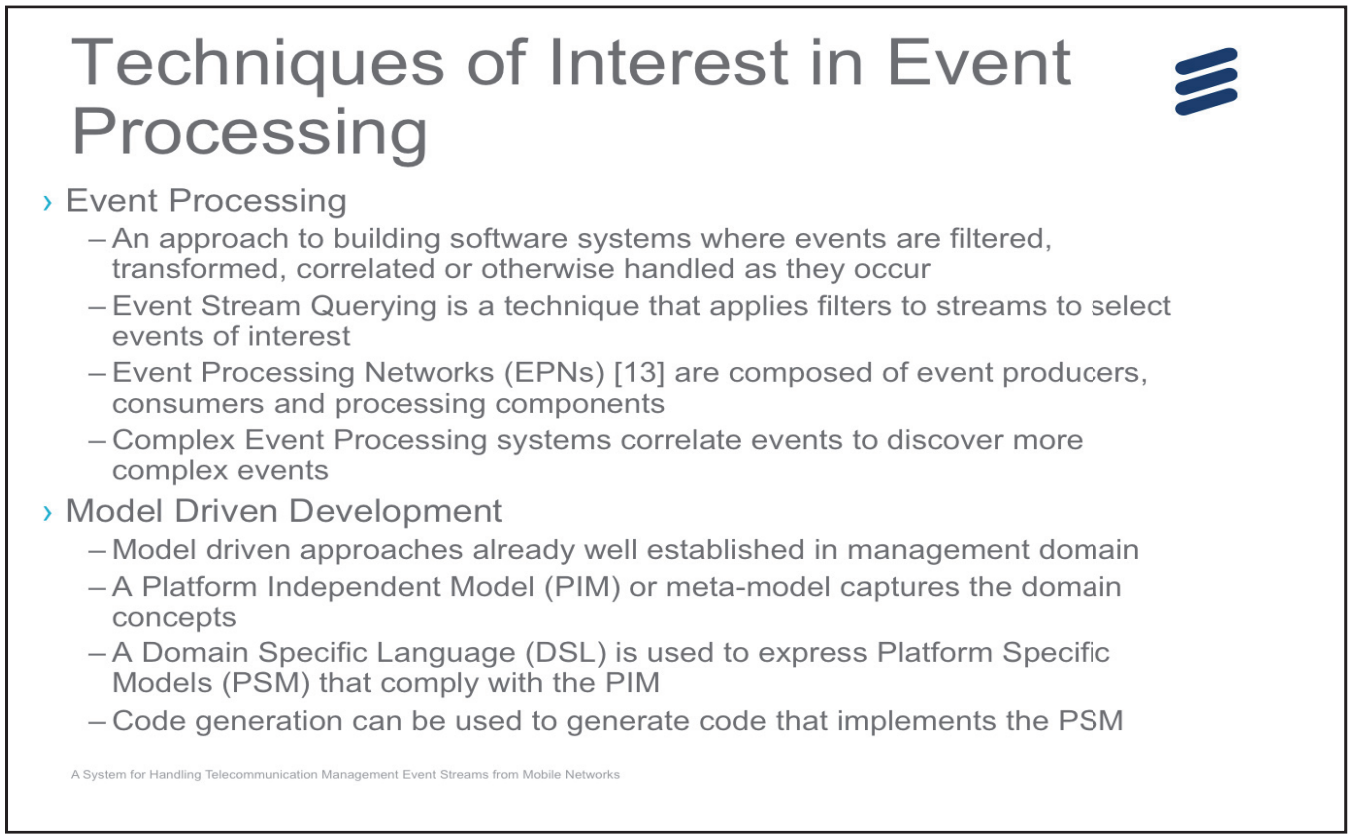
Network elements generally write events to a log file, which is periodically collected from the network element by the management system using the same mechanism as is used for counter files. This approach, with its inherent delays, is being replaced by push mechanisms that use streams. New sources of events such as IPFIX [12] and quality reports from mobile terminals [4] are inherently streamed.

Event handling in the management system must be able to handle the large volume of different events coming from a large number of nodes in real time in a flexible manner. In addition, it must be able to reconstruct event sequences from raw events for analysis and troubleshooting.

The batch processing architecture of current performance management systems is not well suited to handling real time event streams. In current systems, information is usually stored in a data store such as a RDBMS or a system such as Hadoop. Such systems have an inherent ETL (Extract Transform Load) based approach, and correlate data using a batch processing model. It can be a number of minutes before results are available from such systems.

In real time event processing systems, it is more important to process and present as much of the incoming information as possible as quickly as possible; even if a small number of incoming events are dropped. It is more important to have a very accurate picture of the mobile network immediately than a completely accurate picture of the mobile network a few minutes in the future.

It is very important to have a global understanding, common handling, or common transport of events in a management system that is handling events. If the format of input files and streams is hard coded into parsers, data analysers, filters, adapters, and correlations, all of these components will have to to be updated when new events are added or events change, incurring large costs in time and money.



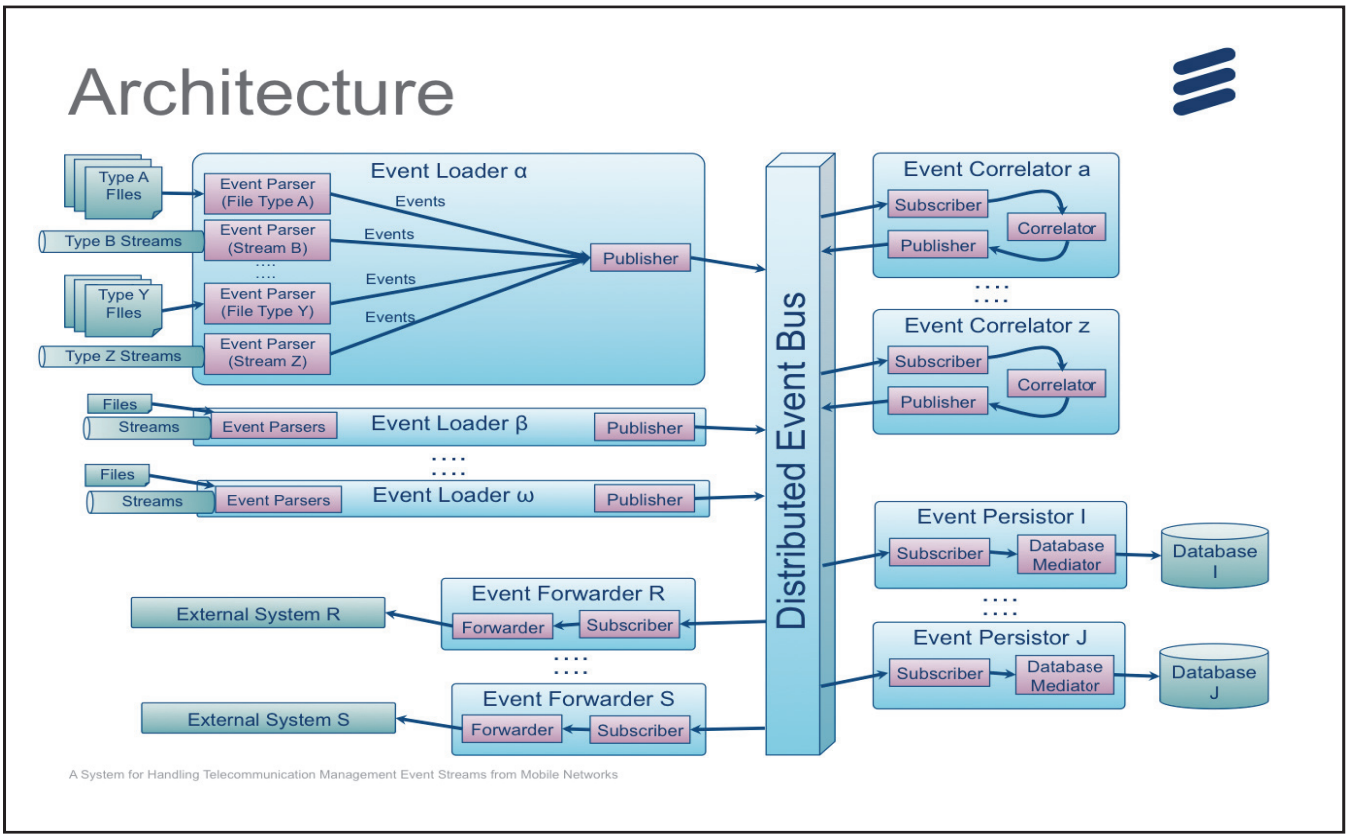
Etzion and Niblett [13] define event processing as computing that reads, creates, transforms, and deletes events. They define areas of application of event processing: *observation, active diagnosis, dynamic operational behaviour, information dissemination,* and *predictive processing*. Event processing in a management system may be applied to all above said five areas. Event processing systems use a pipe and filter architecture in which events are produced, processed, and consumed. In the network management domain, event producers are network elements, other systems such as customer relationship management systems, or the management system itself. Event consumers are management applications and systems, other systems, or the network itself.

Event Stream Querying [14] is a technique where events are passed through a component that allows only events that match a filter through. SQL-like languages can be used to specify filters. Filters can specify event types and attribute ranges that are to be forwarded or blocked. Filters that allow only a percentage of events through or statistical filtering on attribute spreads may also be employed. Such a technique may be employed to in management systems to reduce the volume of events entering the system by filtering out events that are not of interest.

Event producers, processing agents, and consumers connected in a complex manner constitute an Event Processing Network (EPN) [13]. An EPN can handle events going from many producers to many consumers. The components in an EPN communicate using channels to handle component status and context. Event processing in a management system requires a network of event processing elements connected as in an EPN.

Complex Event Processing (CEP) [15] is an approach where events coming from many sources are examined and used to infer another event. CEP platforms are available that consume and analyse event streams to produce new events using a set of queries that define the circumstances in which a new event should be produced. Queries support concepts such as event counts, event sequences, relationships between attributes across events and time windows in which event sequences should happen.

Model Driven Development [16] can be used to specify, implement, and enforce common handling of concepts across a system. In an event processing system, a PIM can be specified as metadata that captures event concepts such as “Event”, “Attribute”, “Attribute Type”, and “Range”. A DSL can be written to enforce compliance with these concepts, for example as an XML schema. PSMs can then be written in XML that specify the actual events that are produced by a network element and that PSM can be used to generate



The architecture of our system is shown in the diagram above.

The system is an Event Processing Network made up of a components that process events, implemented entirely in Java. The components above pass events using a *Distributed Event Bus*, a software framework such as the Java Messaging Service that supports publish-subscribe semantics and distributed event delivery.

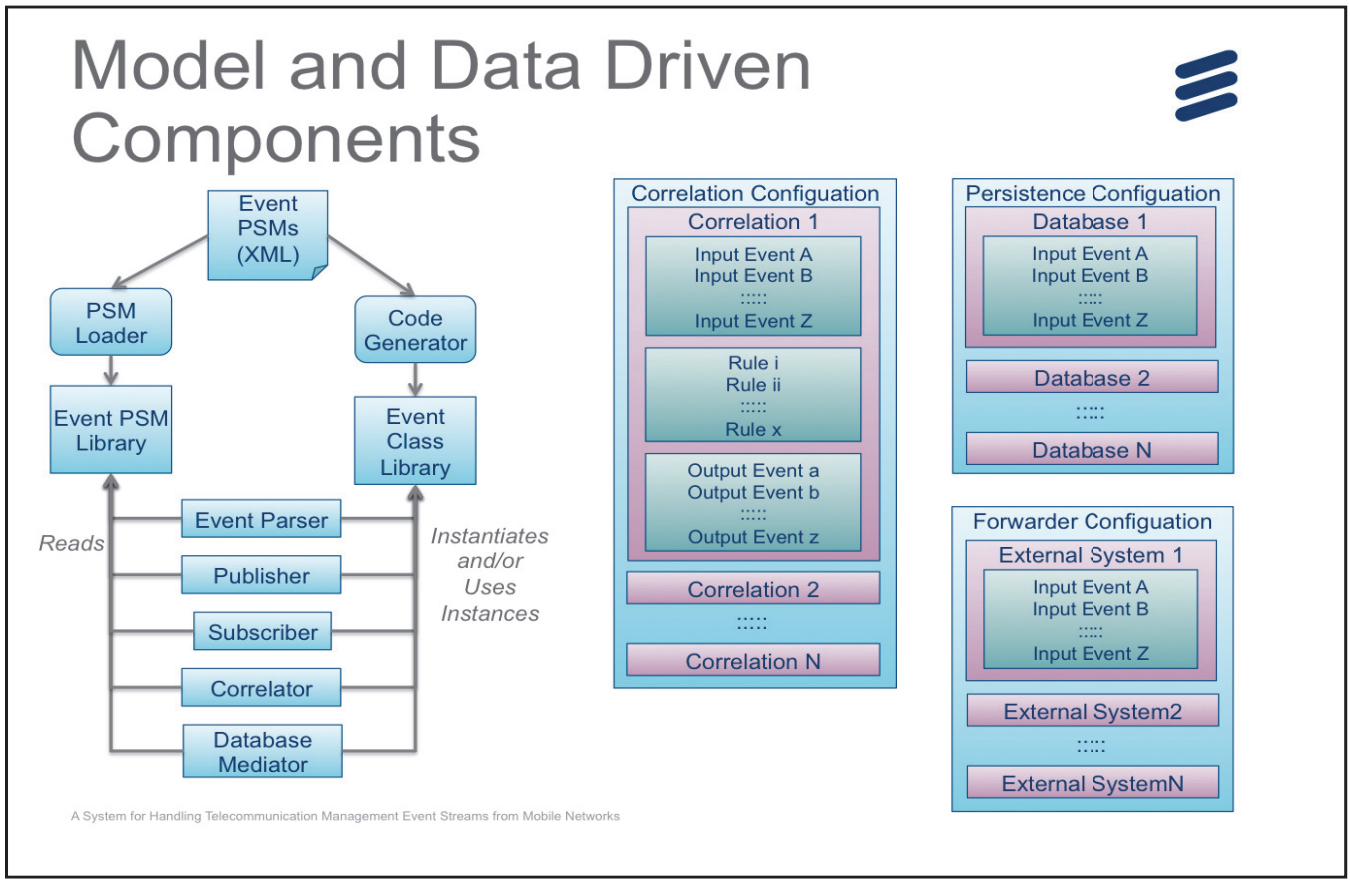
- *Event Loaders* parse incoming events from streams using event definition code generated from models, filer those events, and publish the filtered events onto a distributed event bus. Multiple event loaders may be deployed, with the parsing load distributed across those instances. A loader may be dimensioned to handle events from a set of network elements, a type of event file, or type of event stream.
- *Event Correlators* subscribe for incoming events and use complex event processing to infer mode complex events using queries running a CEP engine. Multiple correlators may be deployed, with each correlator handling a single correlation. Correlators may be nested, a correlation may use an event output by another correlation as one if its input events.
- *Event Persistors* subscribe for events and store then to a database. A persistor instance is deployed for each database system being used. Database schemas are generated automatically from event models.
- *Event Forwarders* subscribe for incoming events and forward those events to other systems, with one or more forwarder instances being deployed for each external system supported.

The system processes events in real time. Events are parsed, correlated, persisted, and forwarded as they come in. The system is horizontally scalable; multiple instances of each component can be deployed in a Java virtual machine (JVM), and multiple JVMs running a set of components can be deployed on one or more hosts.

The system manages its event load in three ways:

1. Event parsers use event stream filters specified in the event DSL to filter only events of interest. Events can be ignored entirely or filters can be applied on event attributes.
2. Publishers only publish events that have been subscribed to onto the distributed event bus. Unsubscribed events are not published.
3. Correlations can be used to aggregate events. If the content of many events can be aggregated, a correlation can be specified for that aggregation. Persistors and forwarders need only handle the aggregated event rather than storing all the incoming events.

In contrast to conventional management systems, the default action is to ignore an event unless some action



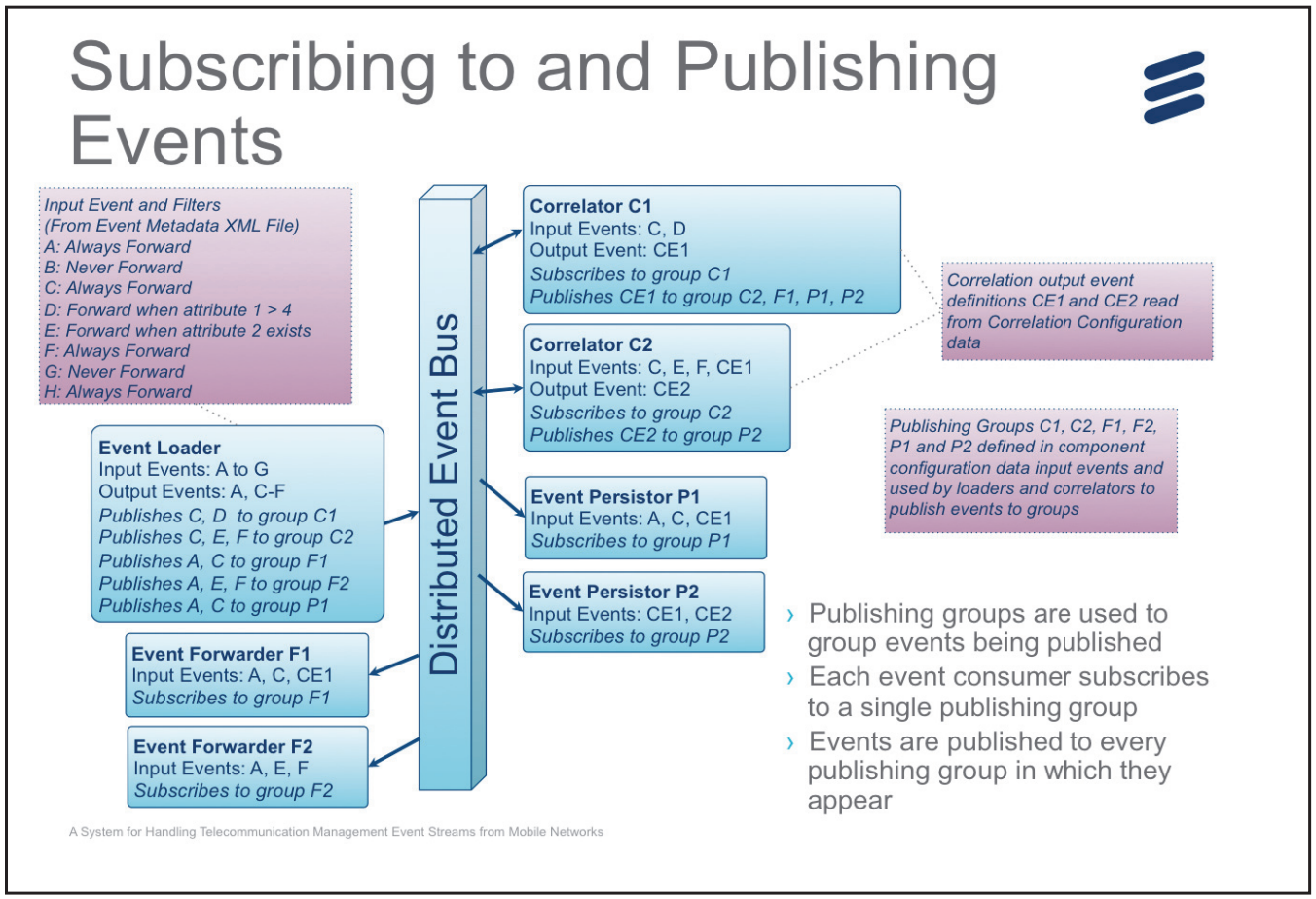
The system uses a Model Driven Development approach for handling events. The PIM for events is specified as an XML schema, which defines an XML Domain Specific Language for specifying events. A PSM is written in XML which contains the event definitions for all events in a domain. PSMs exist for UMTS, LTE, core network, and correlation output events. The structure of each event is defined: an ID, name, and attribute list. The type and range of each attribute is also specified, as are filter rules to be applied to the event. The PSMs for each event are loaded into memory at system start and are available for components to use.

All events are sub classes of a base event class. That base class has some general member variables which all events have such as event ID, name and version. The base class also has virtual methods; each supported input format and output format has a virtual method. For example, if syslog input and CSV output is supported, a virtual method for reading an event from syslog and for writing an event to CSV will exist. A code generator is used to generate a class for every event, which has concrete member variables and methods fore each input and output format.

The system handles event versioning by considering all versions of an event during code generation. An event with member data that is a union of all event versions is generated. Version specific methods are created to read and write specific versions of the event.

Parsers, persistors, and forwarders use code generated classes. A parser for an input format such as syslog reads the event ID from the event header, and looks up the event in the PSM library to get a reference to the Java object that implements the event. An instance of the class is created and the parser calls the appropriate input method implementation to read the event object member data. A persistor for a database is passed an event object, and calls the implementation of its output method to store a class instance to the database. Forwarders use the same mechanism. Publishers, Subscribers, and the Distributed Event Bus are only aware of the event base class and do not examine event-specific member data.

The events consumed by each correlation, forwarder, and persistor are specified and for event publishing as explained in the next slide. For correlations, the queries to be loaded into the CEP engine and the output events generated by each correlation are specified.

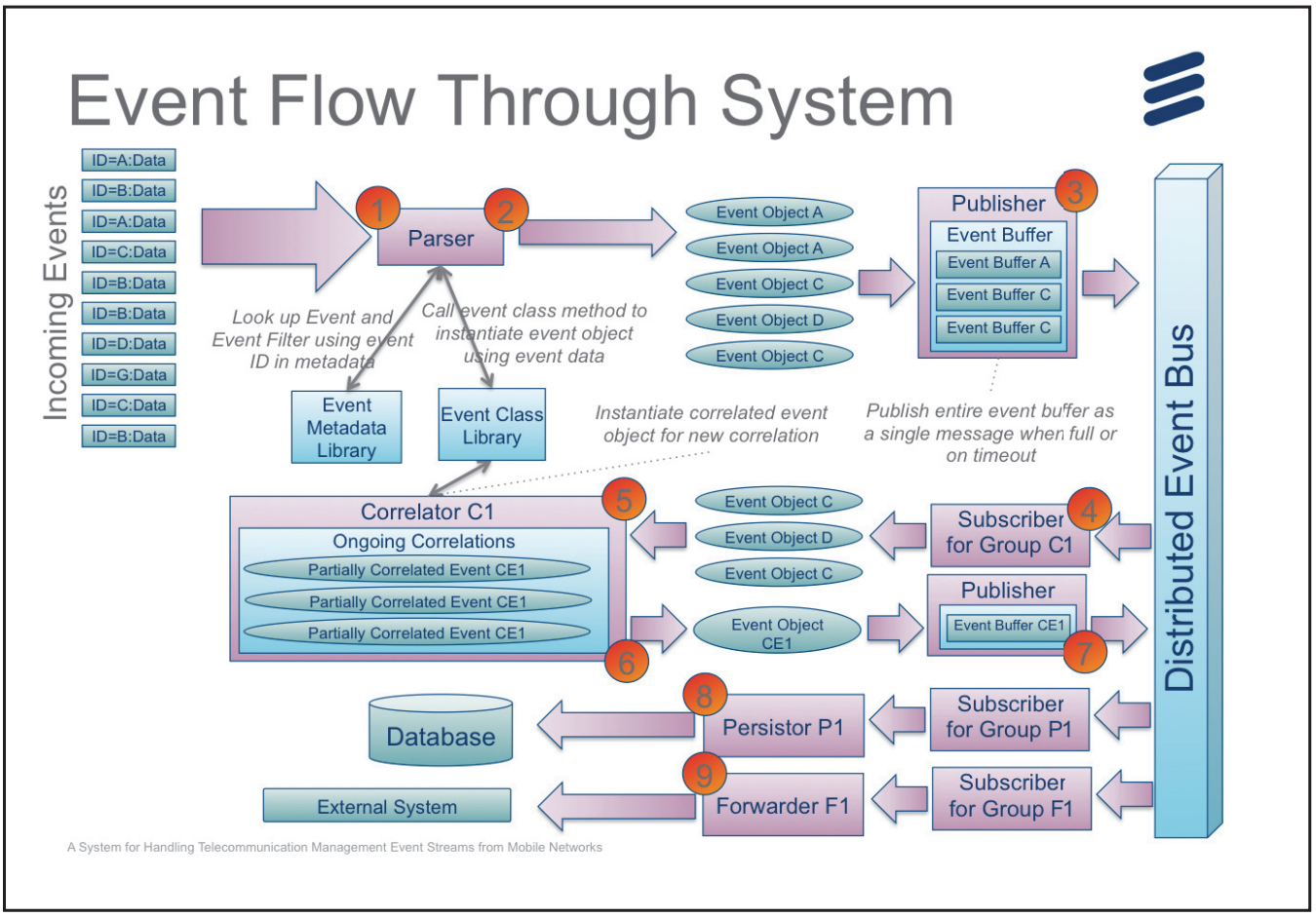


The input events specified for each correlator, persistor, and forwarder are used to build *publishing groups*; correlators, persistors, and forwarders make a single subscription to their publishing group to receive all the events they require. Once the event loader has filtered incoming events, the publisher of the loader uses publishing groups to publish events. An event may be published many times as in the case of event A above, once for every publishing group it appears in. If an event does not appear in any publishing group, it is not published at all, see event H above.

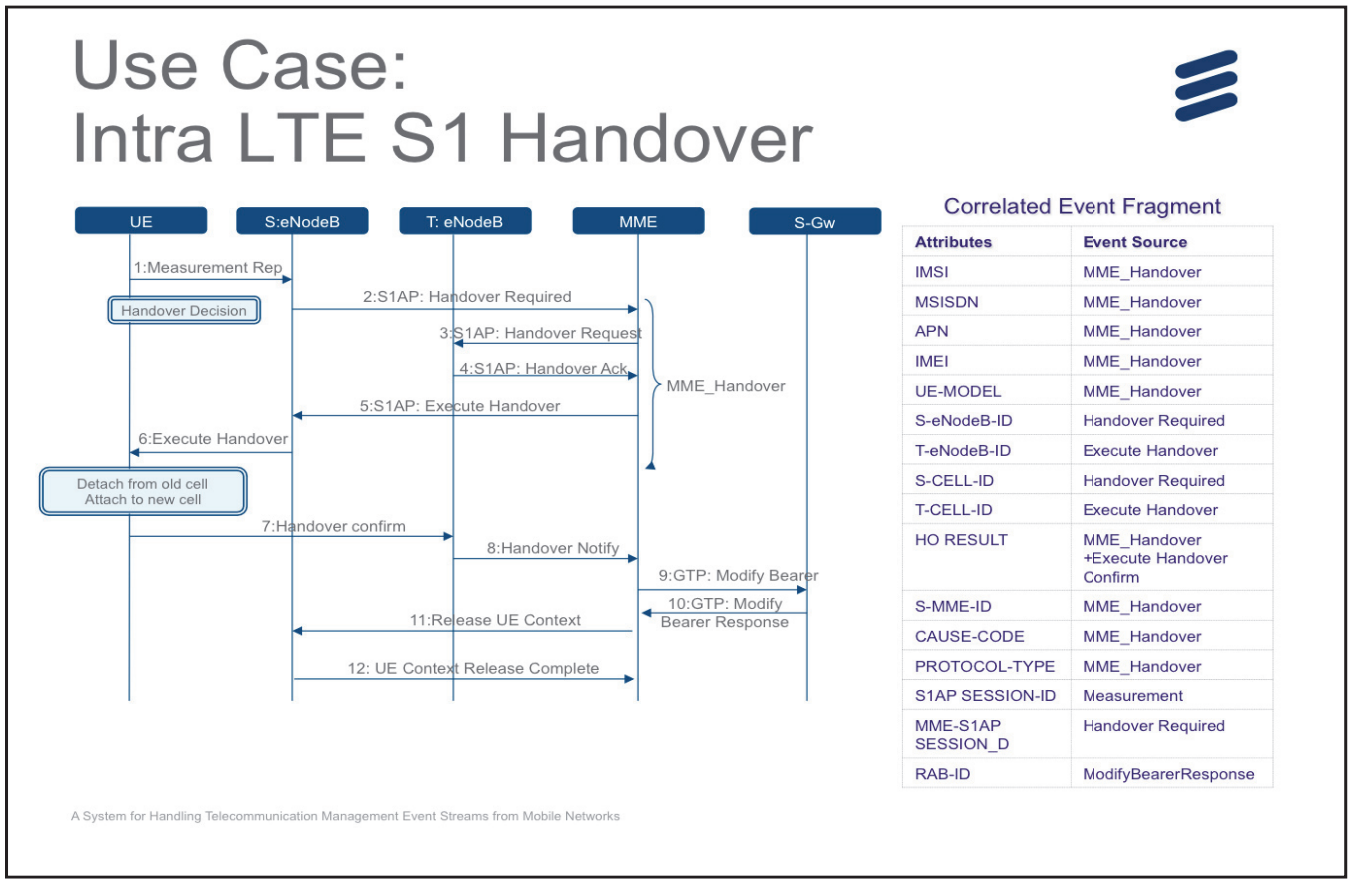
Correlations subscribe to and publish events. Correlations can be nested, a correlation can consume an event produced by another correlation, Correlator C2 consumes event CE1 from Correlator C1 above. Correlators use the same mechanism for publishing as publishers, correlated events are published to all event consumers that have subscribed for a given event.

Using publishing groups allows the type of events that are published to be controlled in an efficient manner. Using publishing groups, it is clear which consumer is consuming each event, and only used events are published. Its drawback is that events may be published multiple times. One alternative less efficient approach is to publish all events to the distributed event bus, letting subscribers subscribe independently to events. Another more efficient but more complex approach is for publishers to maintain a list of all events to be published, and all subscribers to each event. When an event is published, it is tagged with all subscribers that have subscribed to it. Subscribers read events that carry their tag. In our experience, the publishing group method works well, and we have not seen significant load increases due to duplication of events.

A buffering mechanism is used to increase the efficiency of event publishing. It is much more efficient to publish a small number of large messages onto the Distributed Event Bus that to publish a large number of small messages. Publishers hold event buffers, one for each event type. When an event is received by a publisher, it is not published immediately but is held in the event buffer. When the event buffer fills, all the events in the buffer are sent at once. A timer, set to a low number of seconds, periodically flushes all buffers so that infrequent events are not unduly delayed by the buffering mechanism.



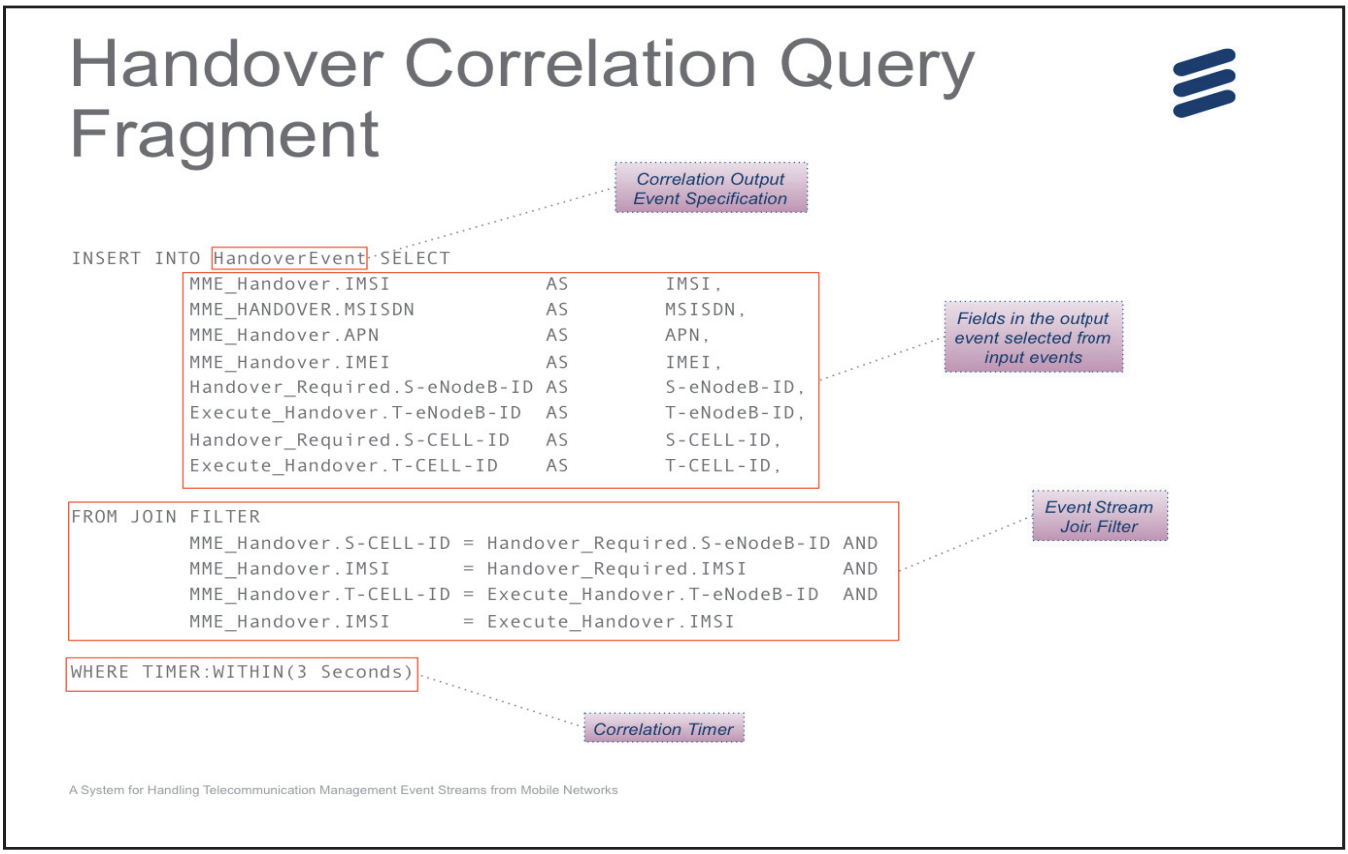
1. A set of events is received by the system. The parser reads the header of every event to find the event ID. It looks up the event filters for each event. It drops all occurrences of events B and G.
2. The parser gets a reference for the generated java class for events A, C, and D, instantiates objects, and calls the appropriate concrete input method to read the event data into the object member data.
3. The publisher takes the incoming events, and buffers them in event buffers. When an event buffer fills or times out, the event buffer is published to the Distributed Event Bus.
4. The subscriber for Correlation C1 receives events of type C and D and passes them into the correlation engine.
5. The correlator has a number of ongoing correlations. It uses its correlation queries to decide which partial correlation the three incoming events should be consumed by.
6. One correlation completes and a correlated event of type CE1 is produced and forwarded to the correlation publisher.
7. The correlation publisher uses the same procedure as in 3. above to publish the correlated event.
8. Persistor P1 received the events to which it has subscribed, calls the appropriate generated output method on the incoming Java class and persists the event.
9. Forwarder F1 received the events to which it has subscribed, calls the appropriate generated output method on the incoming Java to transform the event into the format required by the interface to the external system and forwards the event.



The sequence above is an extract from a realization of an actual deployed use case.

The control sequence above occurs in a LTE network when User Equipment (UE) is handed over from a cell on one eNodeB to a cell on another eNodeB. See 3GPP TS 23.401 [17] for more details. The control sequence is simplified due to space limitations.

1.The source eNodeB decides to initiate an S1-based handover to the target eNodeB, as a result of a measurement report indicating a better cell is available.
2.The source eNodeB sends Handover Required to the source MME, indicating which bearers are subject to data forwarding. The source MME selects a target MME which may be the source MME.
3.The Target MME sends Handover Request to the target eNodeB. This message creates the UE context in the target eNodeB, including information about the bearers, and the security context.
4.The Target MME receives the response from the target eNodeB.
5.The source MME sends Execute Handover to the source eNodeB, specifying a Target to Source transparent container, bearers subject to forwarding, and bearers to release.
6.The eNodeB uses the Target to Source transparent container and forwards the Execute Handover command to the UE, initiating a synchronization process between the UE and the target eNodeB.
7.After the UE has successfully synchronized to the target cell, it sends Handover Confirm to the target eNodeB. Downlink packets forwarded from the source eNodeB are sent to the UE. Also, uplink packets can be sent from the UE, which are forwarded to the target Serving GW and on to the PDN GW.
8.The target eNodeB sends a Handover Notify message to the target MME.
9.The Target MME sends a Modify Bearer Request message to the target Serving GW for each PDN connection, including the PDN connections to be released. Location and time zone information is included if requested.
10.The target Serving GW sends a Modify Bearer Response message to the target MME. Then the Target MME executes the tracking area update procedure (not included in the sequence diagram).
11.The Source MME sends Release UE Context message to the source eNodeB.
12.The source eNodeB releases its resources related to the UE and responds with a UE Context Release Complete message.



The correlation query fragment shown above is part of a query that correlates a LTE handover. Query scripts are composed manually, and are deployed on the CEP engine at run time. The CEP engine begins to filter events using the script immediately, without any need to restart or reset software components. The fragment is selected to illustrate the most important concepts used in building such a query. The query language resembles SQL and many of the keywords are the same as those used in SQL. The four highlighted elements of this query are explained below.

The target of the query is the first element of the query. It is the event that will be produced by the correlation. In this case an event called *HandOverEvent* will be produced.

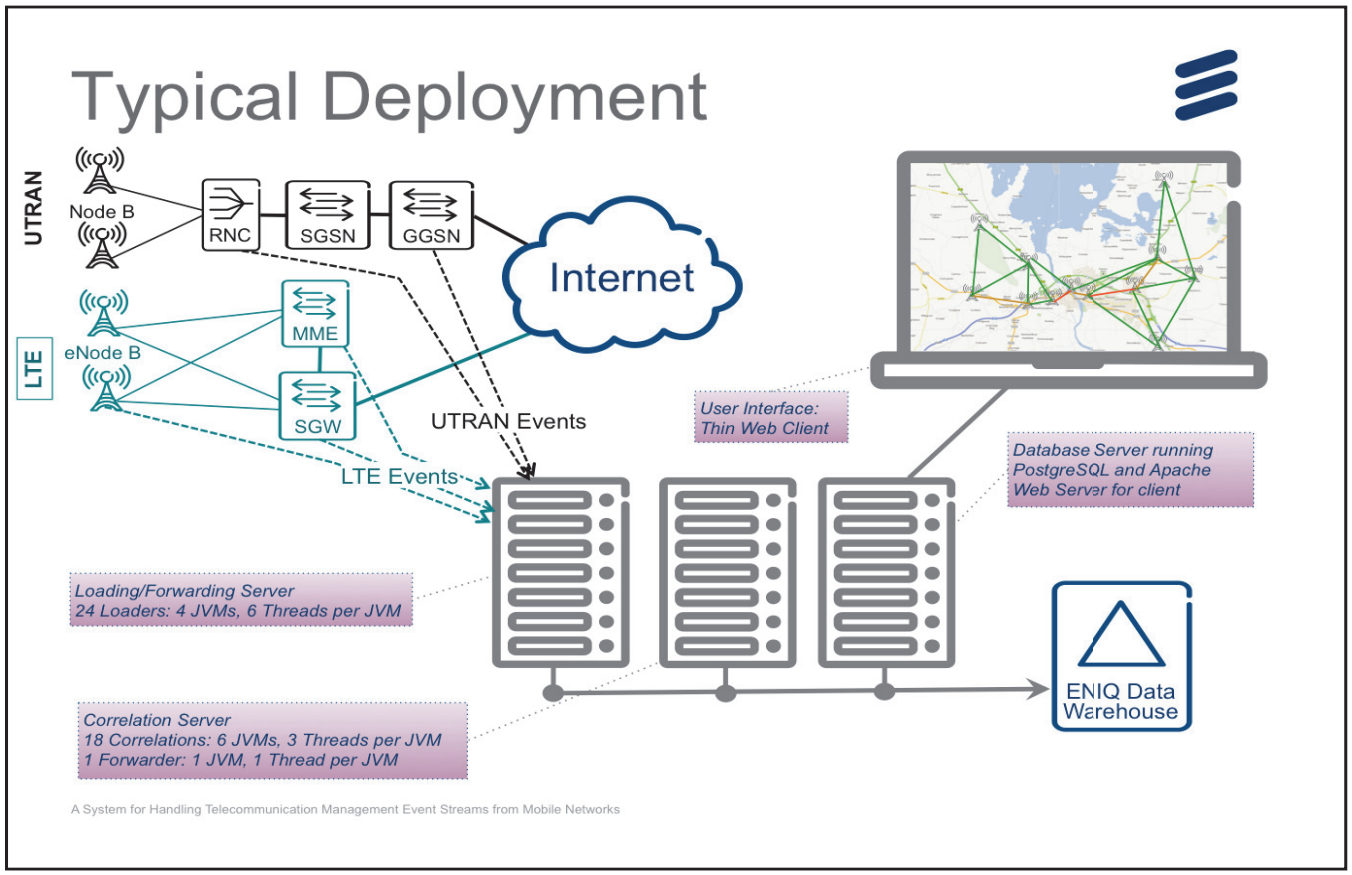
The Select part of the query specifies what fields from matched events should be inserted into the target event of the quey. The IMSI is inserted from an internal event called *Internal_IMSI*, the UE model, source, and destination eNodeB IDs are read from the *MME_Handover* event and so fourth for other fields read from other events.

The *Join Filter* is the third element of the query. It specifies the criteria to be used for matching events, a concept similar to a Join opertation in SQL. This filter specifies that *MME_Handover, Handover_Required*, and *Execute_Handover* events should be matched when the *IMSI*, *S-CELL-ID*, and *T-CELL_ID* fields of those events match. Any event of these types where the fields match as specified in the filter will be considered in the correlation. Much more complex filter criteria can be specified if required.

The final element of the query is a correlation timer. It is the time window over which a correlation will be considered. The time window starts when the first event that matches a filter is received by the correlation. Correlation timeout actions may be specified on a correlation; a partially correlated event or error event may be produced, an error may be logged, or no action may be taken.

CEP engines have very feature rich languages; for example procedures and functions are supported, and utilities for handling time and arithmetic operations are included. The reader is referred to the Wikipedia page for Complex Event Processing[1] for a list of well known CEP engines.

The great advantage of this approach over current systems is that results are available in real time, in contrast with current management systems which generally perform analysis using ETL-based approaches on historical data.



A proof of concept implementation of the system has been developed, verified, and deployed in some customer networks.

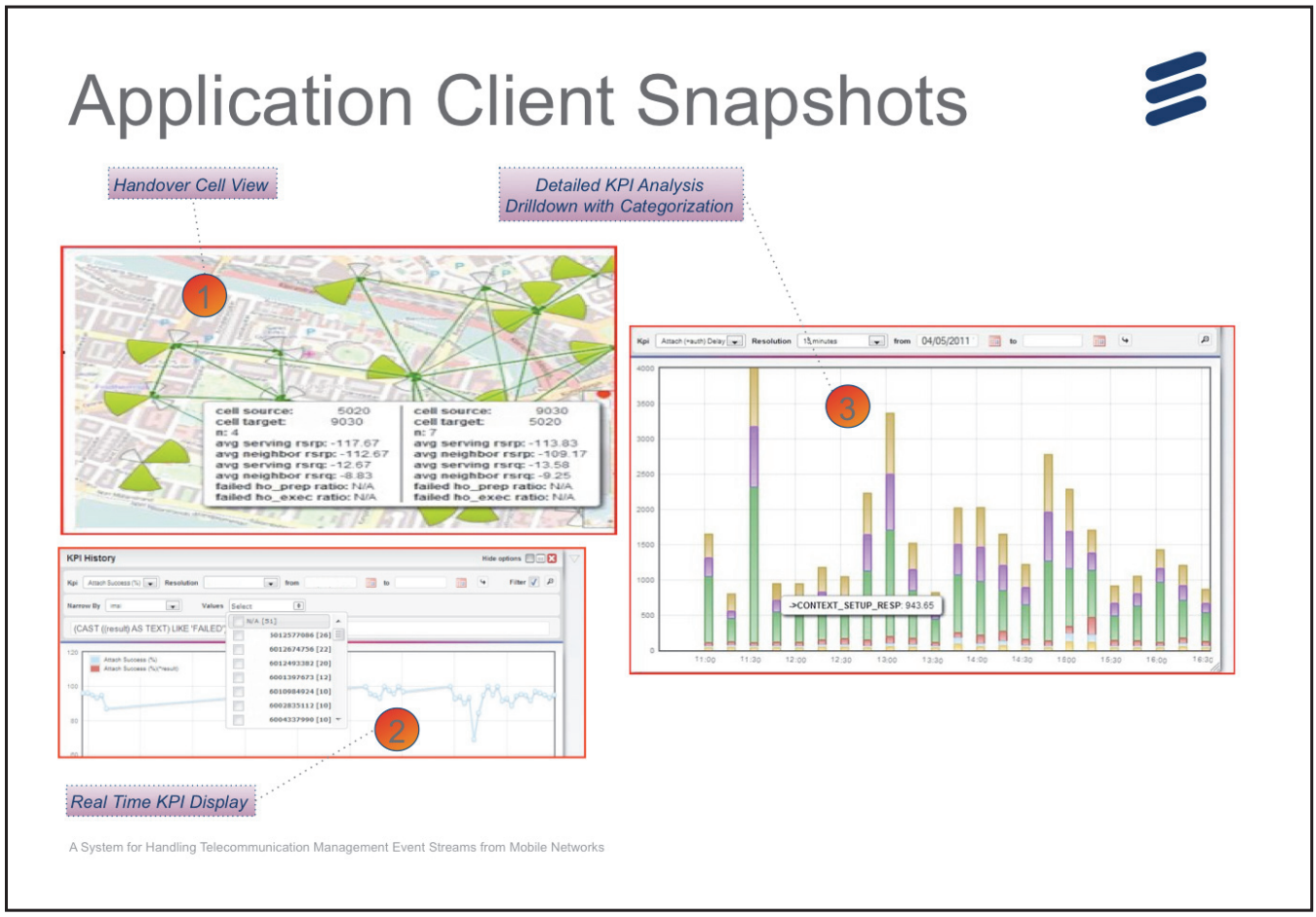
The diagram above shows our a typical deployment of the proof of concept system on three nodes, handling a medium sized mobile network. The hosts may run on standalone machines or on blades. The first host is a loading/forwarding server, the second is a correlation server, and the third is a database server. All servers run run Red Hat Enterprise Linux (RHEL) 6.3, with a PostgreSQL database used as a persistent store.

The loading/forwarding and correlation servers are configured with more memory than the database server because loading and correlation is memory intensive. Data is not stored on these hosts so disk space usage is only what is required by the operating system and application software. The database server, has large disks to hold events that have been persisted. A client application, described on the next slide, was developed to allow monitoring of mobile networks. The client is deployed in Tomcat on an Apache web server that runs on the database server machine.

In the proof of concept system deployments, events are received from eNodeB, MME, and SGW nodes in LTE networks and from RNC and GGSN nodes in WCDMA networks. The number of JVMs used for loading depends on the size of the network that is reporting events. The optimum number of loading threads per JVM is six, when more than 6 loading threads are used per JVM, the overall throughput of events on that JVM declines. A forwarder running in one thread in a single JVM forwarded events to the Ericsson Network IQ data warehouse. The forwarder runs on the same host as the loader because the bulk of the events to be forwarded originate on the loader.

A total of 18 distributed correlation processes were utilized to produce the correlated events for the client application and for forwarding. Three correlations run in each correlation JVM. Correlations require much more resources than loaders for a given event load because of the much higher degree of complexity inherent in executing correlation queries.

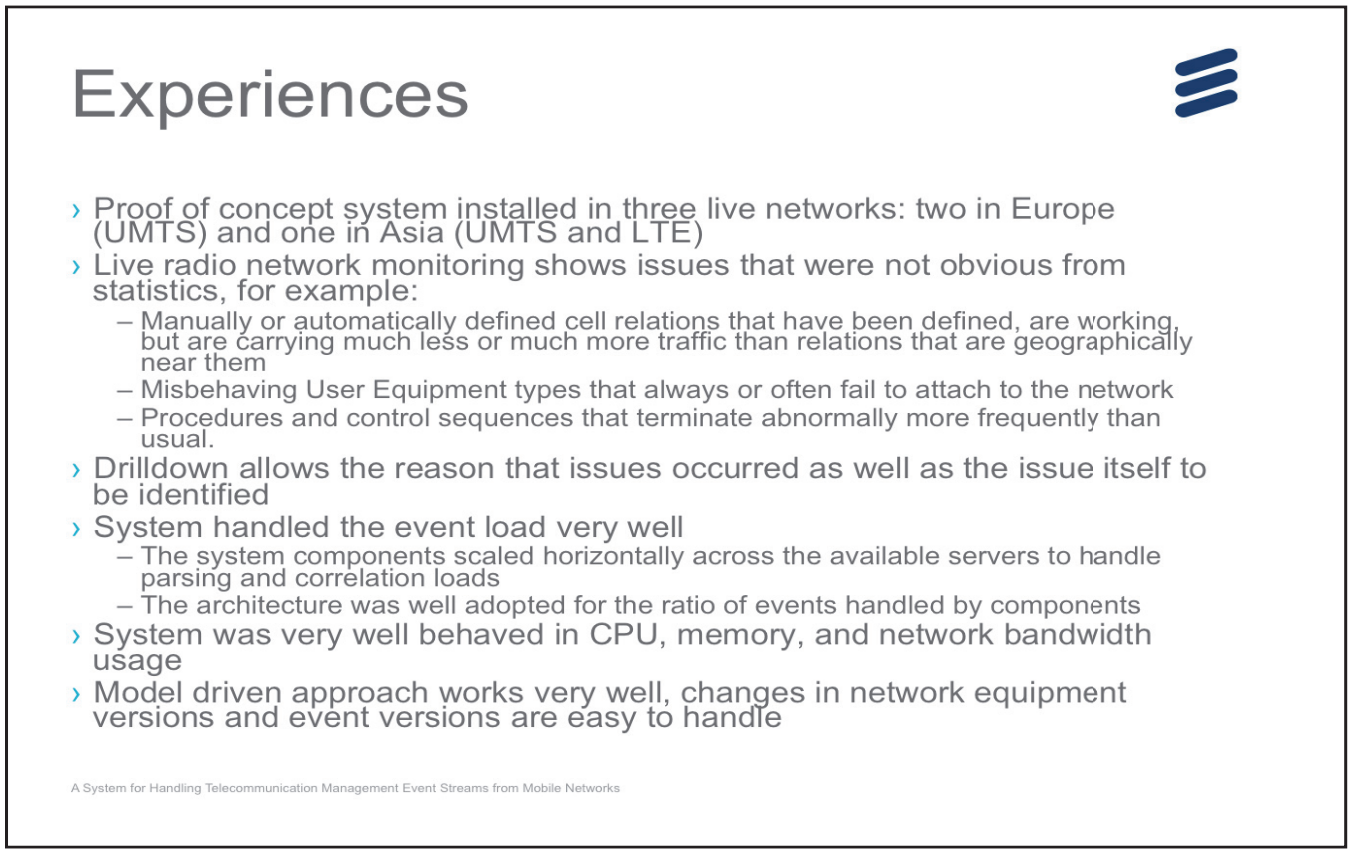
The bandwidth requirements for streaming are less demanding than for a batch file based solution because a low volume of events are sent continuously rather than all events sent together in a file at the end of reporting periods.



A client application was developed that uses raw and correlated event data to display the status of mobile networks in real time. The client shows a number of aggregated views of the network, supports filtering of views using parameters such as Network identity, Node, Cell ID, IMSI, and terminal type. The client allows drill-down on all views, at the lowest level, users can drill down to see actual the raw or correlated events.

The three screen shots above show some of the features available in the application and have been selected to demonstrate some aspects of how event based monitoring can be used.

1. This window shows the handover relations for a selected geographical area in real time. The green arrows indicate the directions of handovers that are occurring. Selecting an arrow displays detailed metrics on the handovers between two cells, selecting a cell displays detailed metrics on the performance of cell. Further filtering and drill down can be executed from either detailed metrics menu.
2. KPI charts for a selected time period can be displayed such as attach success % shown in this window. The window is updated in real time. KPIs can be filtered geographically or using parameters such as terminal type or IMSI as shown in this screenshot. An IMSI filter displays the KPI for an individual subscriber. Again, drill down can be executed from this window.
3. The third window shows an example of a drilldown analysis of a KPI. Drilldown analysis windows are specific to the KPI in question. This particular drilldown analysis shows an analysis of average Attach procedure times in 15-minute buckets for the last 5.5 hours. The colours on the bars indicate the portion of the overall average Attach time attributable to certain parts of the network.



We have installed the proof of concept system in three networks. Network operations staff found the network performance live views very useful. The system was very useful in identifying problems that were not obvious from counters. The ability to show procedures such as handovers in a network context rather than at node level helps in identifying problems and in showing where network optimization efforts might be fruitful The ability to drill down to individual sessions and events eases troubleshooting tasks substantially.

The system architecture performed well under event loads, with event loads distributed well through the system. Information was used from both raw and correlated events in the application developed in the proof of concept. The nodes produced the events required for the proof of concept application as well as events not required by the application. We examined the proportion of the event load borne by each component for this application. Other applications may exhibit different characteristics.

•The parsers consumed 100% and forwarded approximately 20% of the event load. Approximately 80% of the event load was filtered out by event filtering in the parsers.

•The correlators consumed approximately 4% of the event load or 20% of the 20% of raw events that are of interest to the system.

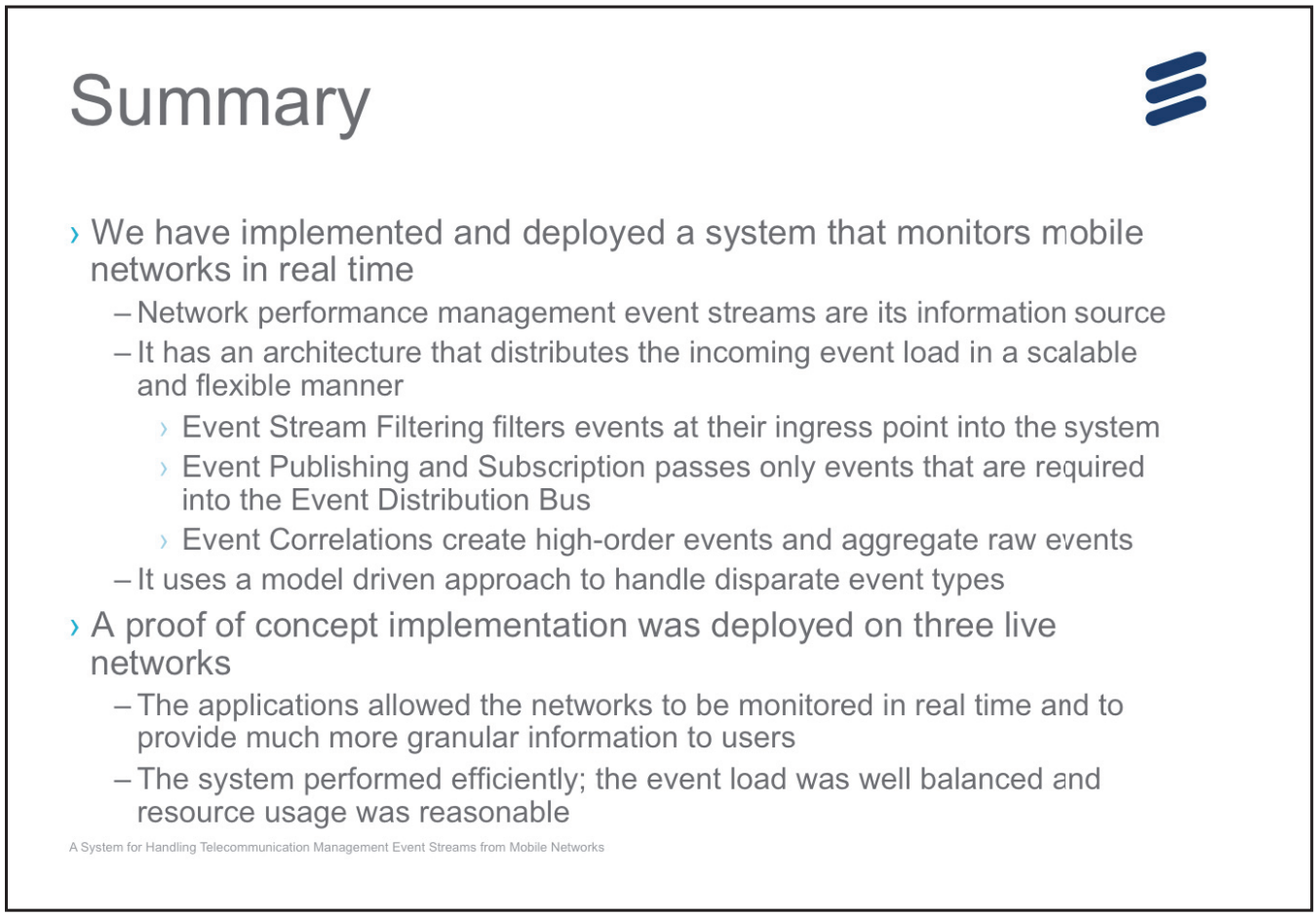
•Correlations produced very different numbers of events per input event depending on the type of correlation. Some simple relationship correlations had a ratio of 1:3 for events produced per events consumed. The handover correlation explained earlier has a relationship of 1:12. Correlations that carry out aggregations can have production ratios of 1:40 or more.

•Event forwarders and persistors consumed the 20% raw event load plus the correlated event load, which was less than 1% of the input event load.

The architecture of the system works well because the simple and fast parsers bear most of the load, and the complex and resource hungry correlators only absorb the event load that they need.

Using a model driven approach has proven to be very flexible. It is very easy to introduce events from different sources and different versions of network elements can be handled together. The distribution and scalability of the system is aided because all components have a common understanding and common handling of events.

Events vary in size between 512 bytes and 1 KB and each procedure uses between 10 and 15 events. The streaming bandwidth requirement is therefore between 40 Kbits and 120 Kbits per procedure.



The system is an Event Processing Network that consumes Events from network elements to allow mobile network to be monitored in real time for Performance Management of mobile networks.

The architecture of the system is highly distributable; each component can be configured to run on a number of hosts or on a single host and uses publish-subscribe semantics and event serialization to distribute events between hosts. All communication between components is carried out using event objects.

The system uses a model driven development approach. The entire system uses models to give a common understanding of the events being processed and produced in the system. The set of events being handled by the system is configurable and can be updated without any software updates. All components in the system have access to and use a common model library that describes the events handled by the system; no event information is hard coded into components.

Event data filtering and parsing into event objects is efficient. Events are filtered at their ingress point, and specific event data loader methods are code generated for each input type. Event publishing is efficient because only events that are used are published, and event buffering is used to ensure efficient use of the underlying messaging infrastructure. Event correlation is configurable at run time, correlation input events, output events, and rules are loaded from configuration files.

A proof of concept implementation of the system has been deployed in three live networks. Those deployment have shown that the system runs well in real network conditions. The real time mobile network monitoring application has been shown to be very useful in finding and diagnosing issues in monitored networks.

This PoC software was installed in a medium size network in Asia, a medium size network in Europe and a large network in the USA. Handover failure analysis, UE session tracking and analysis, attach failure analysis, and paging analysis was evaluated in these installations.

References

1. TM-Forum. SLA Management Handbook, Release 3.0. GB917, May 2010.
2. NGMN. Use Cases related to Self Organising Network, Overall Description. Deliverable, May 2007.
3. P. Gustas, P. Magnusson, J. Oom, and N. Storm. Real-time performance monitoring and optimization of cellular systems. Ericsson Review, (1):4–13, 2002.
4. 3GPP. Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs. 3GPP TS 26.234, December 2010.
5. Ericsson. Mobile data traffic growth doubled over one year. Press Release, October 2011.
6. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016. White Paper, February 2012.
7. 3GPP. LTE: THE Mobile Broadband Standard. Web Page, September 2012.
8. 3GPP. Telecommunication Management; Automatic Neighbour Relation (ANR) Management; Concepts and Requirements. 3GPP TS 32.511, December 2009.
9. 3GPP. Performance measurements: Evolved Universal Terrestrial Radio Access Network (E-UTRAN). 3GPP TS 32.425, June 2012.
10. A. Clemm. Network Management Fundamentals. Cisco Press, 2006.
11. TM Forum. Business Process Framework Suite (eTOM), Release 8.1. GB921, (GB 921), March 2010.
12. E. B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, January 2008.
13. O. Etzion and P. Niblett, Event Processing in Action. Greenwich, CT, USA: Manning Publications Co., 1st ed., 2010
14. J. Hyde, “Data in flight”, Commun. ACM, vol. 53, pp. 48–52, January 2010
15. D. C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001
16. T. Stahl, M. Voelter, and K. Czarnecki. Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons, 2006.
17. 3GPP. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. 3GPP TS 23.401, June 2012.