Objective:

Produce an infrastructure that provides LTE network information in a way that it can be used by operators to make informed decisions.

Target audience:

Designers, Product Owners and Solution Architects with moderate LTE knowledge. Managers of the above looking for an overview.

With this is not:

This proposal provides a mechanism by which LTE based information can be made available to the end user. It does not define or constrain how that information should be prepared or presented, rather, it provides a mechanism that looks after the mechanics so that Solution Architects and Network Engineers can concentrate on adding business value.

Example Use Case:

Local node optimisation. A network engineer wishes to monitor and improve some characteristic of a subset of nodes. They define or reuse a session that tells them what they need to know to measure the behaviour and the system delivers those sessions in the required format to the desired destination for the selected nodes. When they are done, they can tell the system to stop collecting that data, freeing the resources for other uses.

Background and Motivation:

It has long been known that events generated on LTE networks have the potential to provide useful information about the state of the network and its users, and that it should be possible for our customers to utilise this information to monitor and optimise their networks to improve their return on investments. Previous attempts to exploit this knowledge have included ENIQ Events (original and later with tiered storage), CEP (Complex Event Processing), CSL for KDDI/Softbank and the current approach, ASR. In this authors opinion, there are two common factors that led to all the previous attempts being ultimately unsuccessful, and since the current approach used by ASR shares the same characteristics as its predecessors, it is probable that it will have the same outcome.

The first challenge that all these approaches shared was that the volume of input data is enormous while the actually interesting information content is tiny. Common estimates suggest that between 95 and 99.5% of all event traffic is of no interest to anybody and will be dumped sight unseen. The second challenge is that traditional Ericsson design philosophy suggests that if data is available, it must be processed and stored, just in case somebody later finds it useful. These factors combine to produce complex systems that are demanding in terms of their processing and storage requirements resulting in very high total cost of ownership (TCO) but which add only modest value to the available data. This proposal addresses both factors and introduces a number of useful features to increase the value of the information being provided and reduce the cost of providing it.

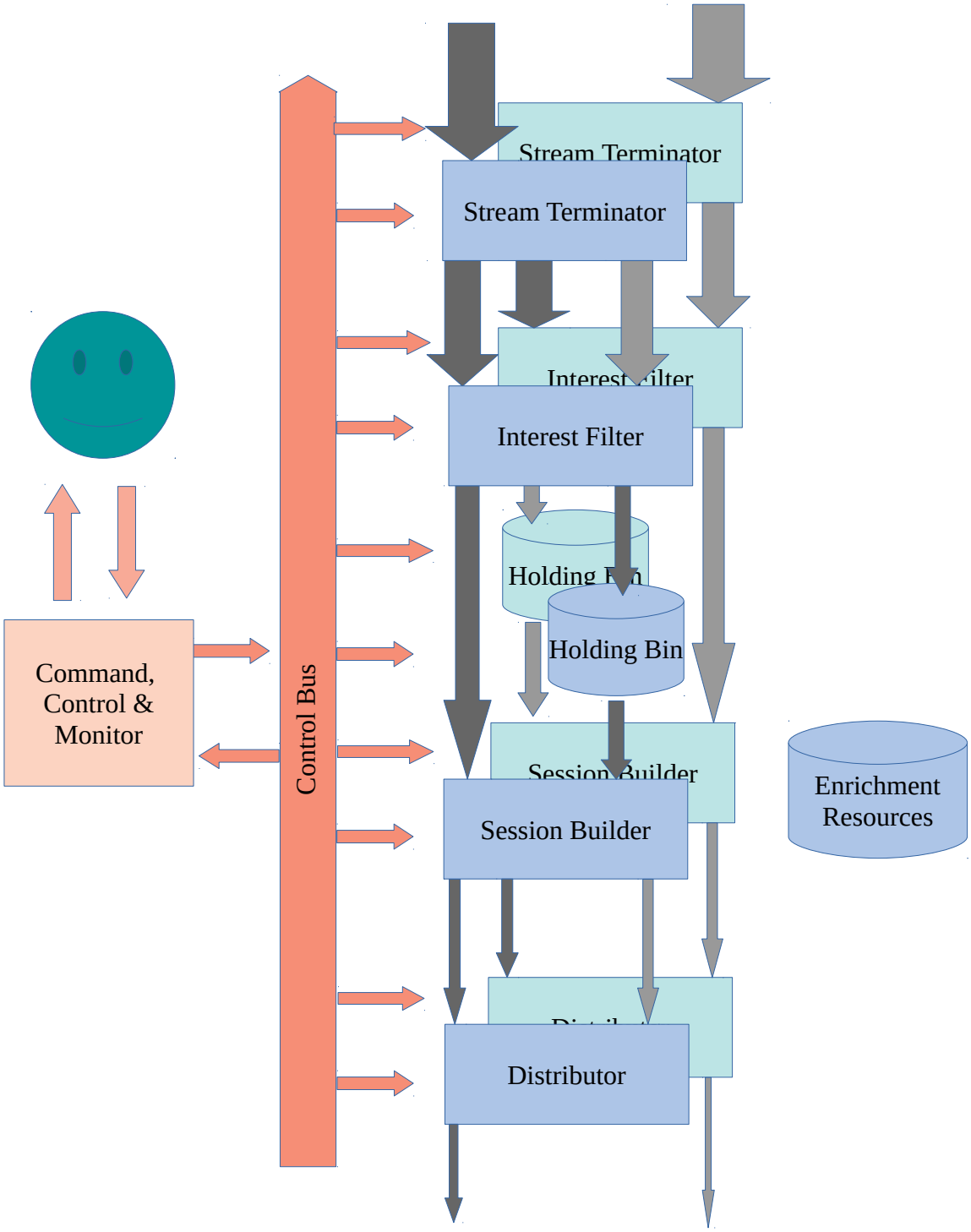
Proposal:

Figure A shows the proposed work flow which uses the well established Model-View-Controller (MVC) design pattern to provide a full feedback loop such that only the data the operator has expressed an interest in consuming needs to be processed, with all uninteresting data being discarded as soon as possible. In this way, the system can be sized for the output desired rather then the traditional approach of sizing for the input available thus producing significant savings in the hardware requirements. This reduces the TCO so helping make the product more appealing to customers.

The other significant difference between this and previous designs is in its approach to scalability. Instead of targeting a fixed load with a fixed traffic pattern on fixed resources, this system is designed such that a given component with given resources can efficiently process a given (quite modest) load and the mechanism of adding additional components to exploit additional resources in a liner manner is documented and tested. This makes the scaling of the system to suit the customers output requirements a simple budgetary and resource allocation decision which the customer can adjust to suit their own needs rather then a software design and deployment issue as is currently the case. A secondary benefit of this scaling approach is that by having no single point of failure, the system has designed in resilience so does not require the duplication of hardware and resources that traditional HA approaches usually demand which increase TCO but do not benefit Ericsson. The third but possibly most important benefit of this approach is that any and every component can be upgraded at runtime without bringing the system down and with minimal loss of data, relieving a major pain point that our customers have been complaining about for many years.

One of the significant advantages of adopting the MVC design pattern is that instead of Ericsson engineers having to decide everything that the end-user might like to see and hard coding that logic into the delivered product, the paying customer gets to decide, and can change their mind at runtime. The Model uses information from the Controller to decide what sessions need to be produced and what they should contain leaving the View to worry about how to format and where to publish the information. If the operator wants to change what they get, they can use the Controller to indicate what they would prefer and it will update the Model and View as appropriate. Obviously Ericsson engineers need to provide the tools and the capabilities that the customer can chose from, but it is no longer necessary to second guess the customers detailed needs or to attempt to provide one-size-fits-all solutions.

Figure A: Proposed work flow



Key components and their characteristics:

Control bus.

This mechanism carries command, control and configuration information from the controller to each component in the system, and monitoring and notification information from each component back to the controller. It is the mechanism that allows each component to do only what is needed, for those needs to be changed on demand and the components to report what they are doing.

Stream Terminator:

One or more Stream Terminators listen for interesting events from one or more nodes on one or more ports. Each is told what events are currently considered interesting through the control bus. They extract the EVENT_PARAM_EVENT_ID from all events received (bytes 4:7), and if it is an interesting Id, it is pushed onto the buses for each relevant Interest Filter for that Id. Stream Terminators report information about their status and resource usage to interested parties by posting to the control bus.

Interest Filter:

These components listen for interesting events from one or more Stream Terminators. They extract the EVENT_PARAM_GLOBAL_CELL_ID (bytes16:20) to identify the node it came from and the EVENT_PARAM_ENBS1APID (bytes 20:23) and the EVENT_PARAM_RAC_UE_REF (34:38) to determine which session it might be relevant to and where appropriate the result code is also extracted. These are used to determine if the event would be of interest to a consumer. Events for which an interested consumer can not be identified are sent to the Holding Bin where the least recently used will be will be held until it is full before being dumped. Notifications about events with non-zero result codes from uninteresting nodes can be pushed to the Control Bus allowing the controller to decide if a Session Builder should take an interest in the relevant node, S1AP Id or RAC_UE_REF.

Holding Bin:

A short term storage facility for events not currently of interest to consumers. When interests change, session builders and other consumers can interrogate the holding bin for recent events that could now be classified as interesting. In this way, sessions prior to and including the trigger event can be built up without the overhead of having to process everything ‘just in case’. Given that 95% of valid sessions last less than 30 seconds, 98% last less than a minute and 99.45% last less than two minutes, the capacity of the holding bin need only be large enough to hold a few of minutes of output from the attached interest filters to be useful. Increasing sizes beyond that will offer diminishing returns and though they can be made as large as the clients budget allows, higher capacities will take more effort and resources to populate, index and purge, so reducing the returns further. The data in the Holding Bin is time relevant data with no recovery value so it should be in memory or on locally attached fast access media. There is no advantage to storing it remotely but there are a number of disadvantages associated with increased latency and IO throughput that would be nice to avoid if possible.

Session Builder:

Standard Session builders listen for events from specific nodes and merge them into Session records as defined by the controller and their internal logic. These modules include auto-generated parsers that know how to decode each event to extract the fields used in the sessions though where possible, they do not decode fields that are not actually used. As a result, Session builders are specific to particular versions of nodes therefore it is necessary to be able to upgrade them in real time to reflect version changes with minimal loss of data. This is accomplished by following the procedure shown in Appendix One. Completed sessions are sent to one or more Distributors which are responsible for formatting and distributing them to the end-user consumers. The Session Builder would also have access to enrichment resources to support adding additional value to the sessions. Different Session builders could have different functionality such as that used by EBS-L where event data is aggregated to produce performance statistics.

Distributor:

These components listen for sessions from one or more Session Builders and publish them to the destinations and in the supported formats that the consumer has requested. Like the others, these components get their configuration information from and have access to the control bus, which also means they can determine the content of the data sessions they are acquiring and so can publish schema information appropriate to the format they are using along with the actual session data. This flexibility means that it is relatively simple to set up a system such that all sessions for a given node are written in one format to a given file, all sessions for a given region are published (in a different format) to a TCP port, and all sessions where a particular error state is detected are published with a different subset of information content in yet another format to yet another destination, which could result in a session for a specific node being sent to three consumers in three different formats.

Command, Control & Monitor:

This component provides the interaction between the system and the operator allowing the operator to monitor what is happening in the system and to change what sessions are produced and what is in them and also to change how they are formatted and distributed. This tool is also used to administer the system.

Implementing Example use case:

Local node optimisation. A network engineer wishes to monitor and improve some characteristic of a subset of nodes. A session is designed to build the data that the engineer needs by selecting the events and their fields and indicating how they should be manipulated. The trigger conditions that start and end sessions are also specified. A Distributor is designed to format the sessions and deliver them to where they are required. Finally, the nodes that make up the subset in question are identified. The controller will take all this information and initiate new Interest Filters for the events involved, Session Builders for the data involved and Distributors for the output produced. The controller then instructs the Stream Terminators covering the nodes involved to feed interesting events to the new Interest Filters, which will feed the Session builders which will feed the Distributors which will supply the network engineer with the information they need to make informed decisions.

Design guidelines.

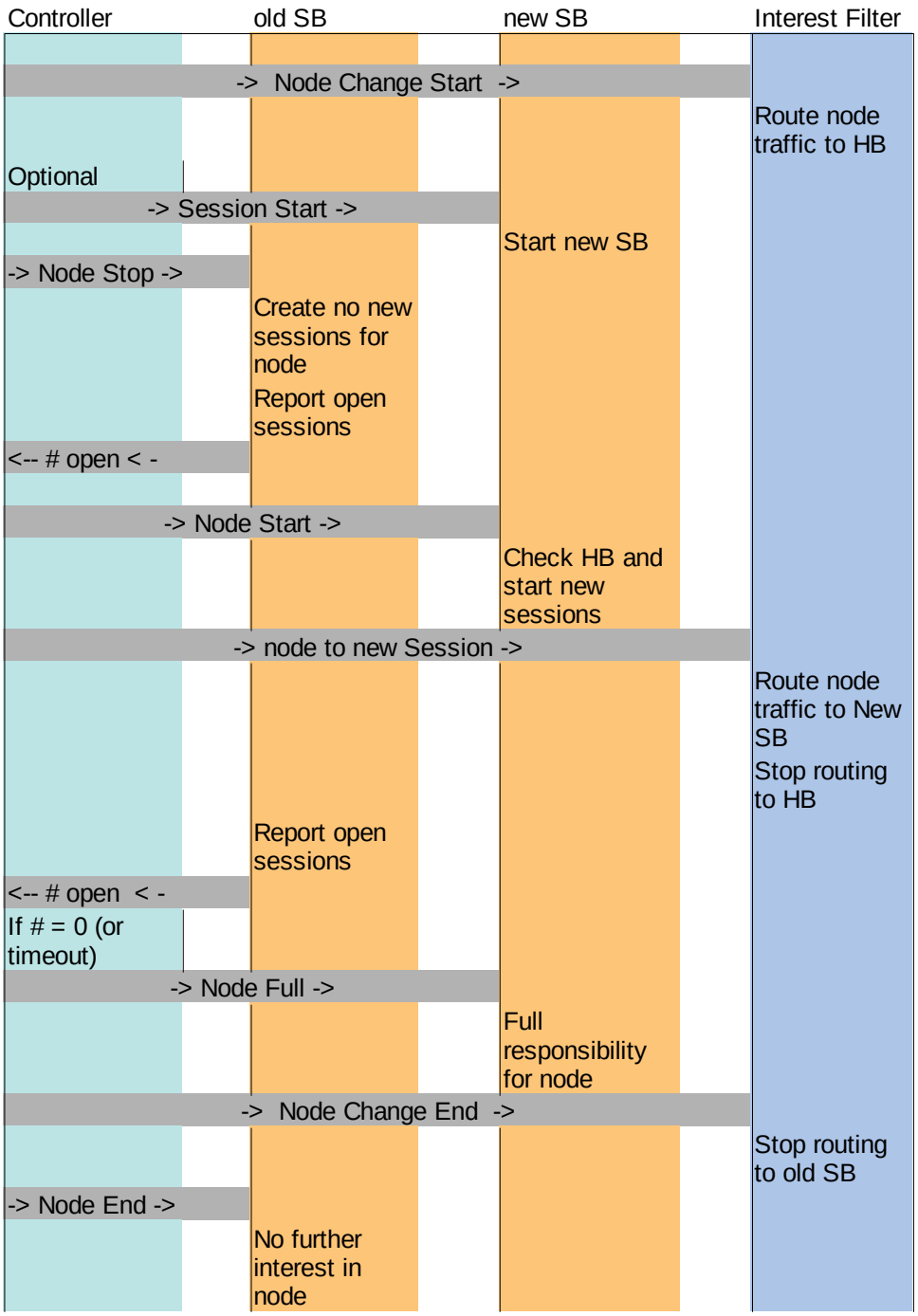
Put very simplistically, all components take their input from one or more sources, apply transforms according to the current (changeable) configuration and write their output to one or more destinations. Command, control and configuration for the component is received via the control bus and progress, status and notifications are sent to the control bus. Additionally, the requirement for resilience means that communications via the control bus will have agreed confirmation, retry and timeout protocols, the exact details of which are not that important, but the result of these multiple parallel IO's is that it imposes constraints on how the components can be coded as they must not block. This means all coders involved in the project must have at least modest familiarity with principles of concurrent programming, while code reviewers should have good competence in this area. The resulting software is more complex but its self contained component based nature actually makes it much easier and cheaper to maintain and reuse, especially since additional or changed functionality can be fully tested in isolation and incorporated into running systems with minimal risk or downtime.

When a component starts, it registers with the control bus to indicate its availability and capabilities (eg, ASR session builder for 17.2 nodes, EBS aggregator for 18.Q3 nodes, Peter's X2 timeout optimisation attempt for 17.1 nodes). The control bus gives the component its initial configuration which indicates what it should listen for and where, what it should do with what it detects, and where it should send its outputs. The component will report its status and progress to the control bus on a regular basis, and may send notifications when particular triggers are detected. It will continue to monitor the control bus for updates to its configuration which it will implement as soon as practical. This commonality of function means that it will be possible to produce a component template that handles the plumbing, messaging and multitasking, allowing solution developers to concentrate on the business logic, reducing development and maintenance costs further. This design approach also has the impact of minimising a components interaction with its environment meaning the component will behave identically however it is run, which has significant benefits during the development and testing life cycle.

Since the nature of the event data being processed is timely with old data having little or no value, backup and restore requirements are therefore reduced to the configuration information managed by the controller component and can be performed without bringing the system down.

A key benefit of this approach is that as the business logic is completely separated from the infrastructure, so changing out one support tool for another just means changing the component template being used which avoids vendor lock-in and makes supporting multiple deployment environments easy. If one customer wants AWS, another wants OpenStack and a third demands a bunch of networked Raspberry Pies, no problem, same application, slightly different component templates.

Appendix One: Real time node version update process



Controller informs Interest Filter that change is coming to one or more nodes.
Interest Filter routes events for affected nodes to Holding bin in addition to current consumers
Controller instantiates new Session builder if required and waits for it to confirm availability
Controller informs old Session builder to stop handling affected nodes
Old Session builder will not create new sessions for existing nodes but will will continue to process existing sessions.

Early Draft – not for sharing. Paul Smith. August 2018

Old session will inform controller of number of open sessions for affected nodes.
Controller will inform new Session to start processing affected nodes from Interest Filter.
New Session will interrogate holding bin for relevant events and start new sessions but ignore sessions not explicitly opened.
Controller will inform Interest Filter that new Session is responsible for affected nodes
Interest Filter adds new Session to consumer list and removes Holding Bin.
When old session reports no open sessions (or after timeout), controller will inform New session that it has full responsibility, and inform Interest Filter to remove old Session as consumer.
New session will now handle session not explicitly opened.
If old Session is not handling any nodes, it can be taken off line and its resources returned to general availability.