A Short History of Complex Event Processing[1]
Part 1: Beginnings

by
David Luckham

First of two articles on the development of complex event processing

Event processing has been going on for more than fifty years. So, you might well ask, what's new about Complex Event Processing? Well, its different from what was going on fifty or thirty or even fifteen years ago. This article is about how event processing has been changing over the years and why.

Remember the old operating systems of the late 1950's and early 1960's? Some of them used "events" to schedule switching between threads of control on single processor systems. Once in a while I still come across someone who thinks that's what I mean when I talk about "event processing" – context switching!
I have to tell them there's a little more to event processing nowadays than there was fifty years ago!

There are four major technology developments over the past fifty years that depend upon various forms of event processing (see figure 1):

- Discrete event simulation,
- Computer networks,
- Active databases,
- Middleware.

Event Processing started with discrete event simulation in the 1950's. The basic idea was that the behavior of a system – be it a hardware design, control system, avionics, factory production line or natural phenomenon like weather – could be modeled by a computer program written in a "simulation language". Given input data, the program would create events that mimicked the interactions between components of the system. Each event happened at a time recorded by a clock. Of course, some events could happen at the same time. But the clock would eventually increase its reading by discrete "ticks" representing the progress of real time. Such models were called discrete event simulations.

[1] © 2007 David C Luckham

The events had the form of messages like “*component C1 created me at time T1 with data values A and B and sent me to components C2 and C3*”. The simulator had to schedule the flow of the events between components in the model, the execution of the components, and the ticking of the clock. Usually the scheduler was nothing more than a round robin queue of events ordered by the clock ticks. When a component was executed it was fed the events waiting to be sent to it, and reacted by creating new events to be sent to other components, and so on. A simulation, when executed on input test data, produced an event trace of the model’s behavior. The final step was to analyze this trace.

Discrete event simulations of this time were certainly doing event processing. Indeed, the models were event driven architectures. The focus, however, was on implementing simulators to schedule the operation of the simulations and distribution of the events as efficiently as possible. One began to see distributed discrete event simulations being run on computing grids, which made the construction of simulators much more interested and complicated - but that’s a separate story. The final area of effort was analyzing the resulting event traces. In the 1990’s research in automated analysis of event traces followed a new path towards introducing more detail about the behavior of the models into the event traces, such as causality between events. The idea was to design the models more precisely so that their simulation would give more detailed information than a simple time ordered event trace. Here was one beginning of CEP. More about this later.

Sometimes a great deal of money rested on the results of discrete event simulations – a new floating point chip design, for example, or a weather prediction for the path of a hurricane. Every major manufacturer had its own discrete event simulation language by the 1960’s. And various general purpose simulation languages began to appear around 1960. Among them we might mention Simscript[2], GPSS[3] and Simula67[4], but there were many, many more. More recently, in the 1980’s and 1990’s hardware description languages such as Verilog[5] and VHDL[6] have become standards supported by a host of commercial discrete event simulators.

[2] http://en.wikipedia.org/wiki/SIMSCRIPT
[3] http://en.wikipedia.org/wiki/GPSS
[4] http://en.wikipedia.org/wiki/Simula67
[5] http://en.wikipedia.org/wiki/Verilog
[6] http://en.wikipedia.org/wiki/Vhdl

Another kind of event processing was involved in the development of computer networks, starting in the late 1960's with the ARPA net. The focus was on making reliable communication between computers across networks by means of events containing sequences of binary data – so called packets. Transmitting or receiving a packet was an event. The basic work involved developing protocols for communicating sequences of packets reliably when the network itself might be unreliable and subject to errors.

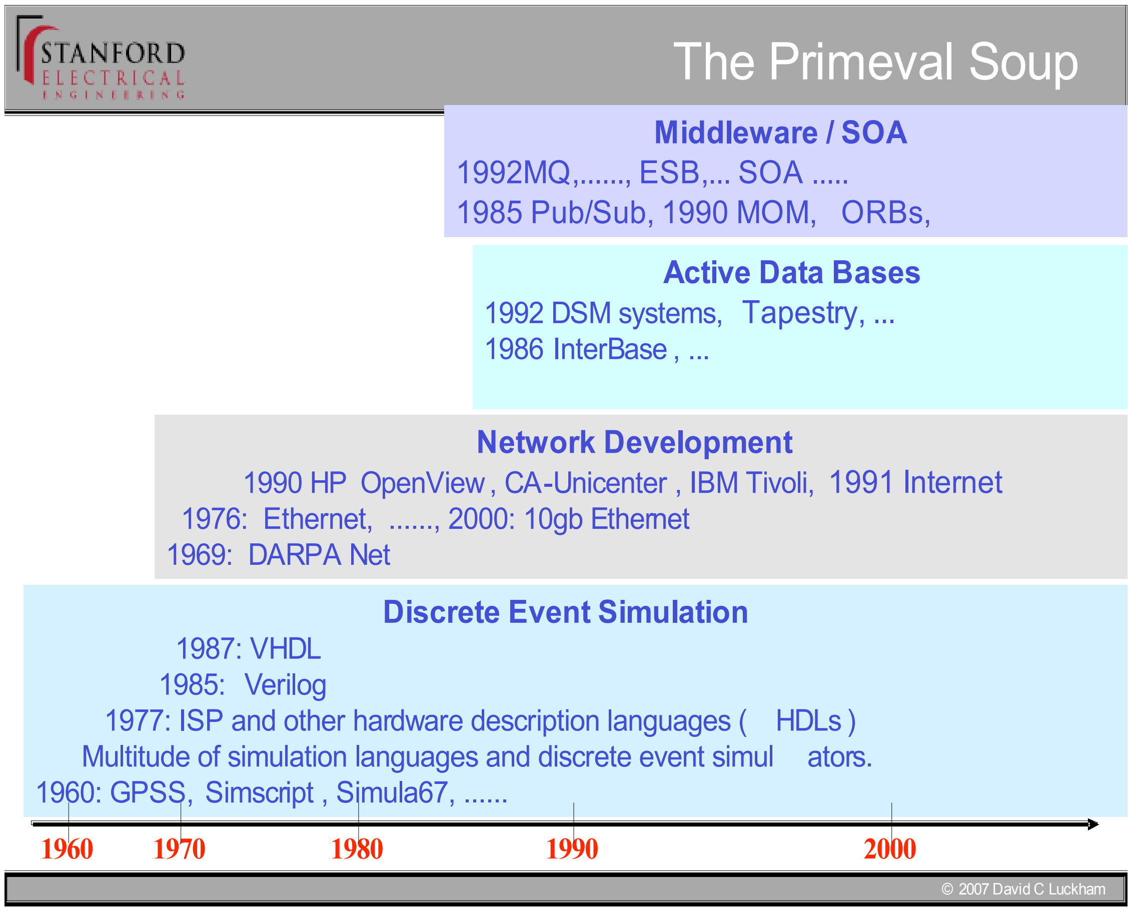
STANFORD ELECTRICAL ENGINEERING
The Primeval Soup
Middleware / SOA
1992MQ,......, ESB,... SOA .....
1985 Pub/Sub, 1990 MOM, ORBs,
Active Data Bases
1992 DSM systems, Tapestry, ...
1986 InterBase , ...
Network Development
1990 HP OpenView , CA-Unicenter , IBM Tivoli, 1991 Internet
1976: Ethernet, ..., 2000: 10gb Ethernet
1969: DARPA Net
Discrete Event Simulation
1987: VHDL
1985: Verilog
1977: ISP and other hardware description languages (HDLs)
Multitude of simulation languages and discrete event simul ators.
1960: GPSS, Simscript , Simula67, ...
1960
1970
1980
1990
2000
© 2007 David C Luckham

Figure 1: Four major areas of Event Processing in the past 50 years.

As this work matured one of the most important outgrowths was the definition of standards for network protocols using the concept of *layers* of events. Two examples are the TCP/IP model[7] and the ISO 7-layer messaging standard[8].

The definition of these layers of events use the concept of *event abstraction*. For example, in the TCP/IP layer, if you think of two computers communicating with one another over a network, you can take an "application-to-application" view. An application on one machine, say a mailer, sends application level events, say email messages, to a mailer on the other machine. So, you think purely in terms of operations on events at that level, *"this message was received from that machine at time XYZ".* That's an abstraction of what is actually happening on the network. At a lower network level, the transmission of a single message is enacted in terms of transmitting lots and lots of binary packets and dealing with timeouts and resends. But the abstraction allows you to forget about all those lower level network details. TCP/IP and ISO 7-layer standards define layers of events from the most abstract (or highest level) application events to the physical layer (or lowest level) electrical events. The definition of each layer describes the events and the operations on them at that layer.

The idea of *event layers* is important in CEP, and is suggestive of the concept of *event hierarchies,* as we shall see.

The 1970's saw rapid developments in networking with the Ethernet (patented by Xerox in 1975) for local area networks. Since then there have been all manner of improvements and extensions through to the 10gb Ethernet around 2000 and, of course, Wi-Fi, and GSM standards for mobile phones. Event processing for communications is big business in today's connected Internet society.

With the advent of networks came a host of network management tools[9] whose job it was to track and trace events and display the results graphically. These tools were the early precursors of BAM (Business Activity Monitoring). They were certainly monitoring events in the network, and sometimes trying to reconstruct the more abstract level activity. So, there in the management of networks you have the tip of the CEP iceberg!

Active Database technology started in the late 1980's as a development of databases to meet the demands of real time processing. Traditional databases were extended with event processing capabilities that allowed them to invoke

[7] http://en.wikipedia.org/wiki/Transmission_Control_Protocol
[8] http://en.wikipedia.org/wiki/OSI_model
[9] see figure 1

applications in response to events. For example, an active inventory database system could initiate a re-order transaction if the inventory fell below a specified level (i.e., an event). To do this, a layer of event processing was implemented on top of a traditional database. It allowed definition of events, simple kinds of event patterns that could be used as triggers of reactive rules called ECA rules (Event-Condition-Action). This layer implemented the matching of the triggers, resolution of conflicts between rules that were triggered by an event, and executing actions such as invoking applications. This in turn depended upon developing languages for specifying events and rules. These languages defined concepts such as composite events, a subclass of complex events, and also temporal constraints.

Active databases provide a platform for some real time applications, for example, workflow execution, movement of data between networks and mobile computers, and a variety of business management applications. An outgrowth of this work is event stream processing, a subset of CEP.

The first middleware company, Teknekron (now Tibco), was founded in 1985. The idea of "middleware" is to place a layer of software "in the middle" between the application layer and the network layer to facilitate application-to-application programming. The middleware hides the underlying networking details. So, a business analyst, for example, can focus on the interactions between applications – so-called services – and program the actual communications by calls to middleware functions provided by an interface.

Probably the first middleware was remote procedure call (RPC), researched in the 1970's. But by the end of the 1980's an array of commercial middleware products were ready to hit the marketplace. Early on were object request brokers (ORBs), and slightly later, message-oriented middleware (MOM). While ORBs were based upon RPC, the MOM approach was decidedly based upon communication by means of events using message queues (MQ) and protocols like Pub/Sub. The publish-subscribe protocol for coordinating requests for service with offers of service was a key paradigm of the MOM movement.

Today, middleware is often synonymous with the concept of an "enterprise service bus" (ESB). There is considerable variation in the philosophy of what an ESB actual is, or contains. Obviously the spirit is that an ESB is a high level application-to-application communications bus, analogous to the hardware bus that carries signals between components in a computer. Alternative views are that it is middleware based upon J2EE or .NET, encapsulating and hiding network details, and offering services like document routing. Some philosophies claim that ESBs contain software products for building service oriented architectures (SOAs). So a commercial ESB offering may bundle together a lot of event driven messaging, often including web services, SOAP and XML messaging. Some ESBs go as far as to say they offer CEP in the form of event interpretation, correlation and pattern matching. Much of this is playing into the currently fashionable market for SOA, with a trend towards EDA.

However, if we ignore the marketing hype, the upshot is that middleware now represents a fourth event processing technology movement. It is based upon event driven communications at a lower level and supports event driven architectures at the application level.

Finally, an apology for all the buzzwords and acronyms!

In the second article we'll discuss how CEP concepts, companies and products are growing out of these four technologies and building upon them to meet the demand to understand the content of the event traffic in today's Internet society.