Thanks to Gartner and Roy Schulte for the opportunity to speak today. I especially want to thank Roy and David Luckham for the excellent presentations they’ve given on Event-Driven Applications. I am going to talk to you about somewhat different aspects of EDA than they did. I hope our collection of presentations is synergistic and helps you to determine whether and how EDA is useful to you.
The business proposition of EDA is that it helps enterprises respond to events. These events may be threats, opportunities or other kinds of situations that require timely responses. Keep these questions about the business proposition in mind: How important is it for your enterprise to respond to situations as they occur? And how much can IT in general, and EDA in particular, help respond in an appropriate and timely fashion?
I’ve got four take-away points and one key question for you. These points and question are in this slide and the next.

The first two points deal with EDA characteristics and the next two with rolling out an event-driven application in your enterprise.

Take Away Point 1: EDA helps your enterprise monitor, correlate and respond to events within and outside your enterprise. For the past 40 years, the focus of enterprise IT has, correctly, been on services within the enterprise. Now enterprises are being increasingly impacted by events outside the enterprise that require timely response. The rise in regulations – e.g., respond to a federal agency within 72 hours if certain situations occur – is one reason that enterprises have to monitor the external world. Rapid reaction to competitors, extended supply chains, and electronic markets are other reasons.

Take Away Point 2: How do you know NOW how your significant others are this very moment? You aren’t hearing from them right now. You may not have heard from them during the break 10 minutes ago. Yet, you are quite confident that you know their current state. Why is that? The reason that you are confident about their state is that the absence of messages conveys information to you. I expect my wife to call me if she runs into an unanticipated situation that requires my response. Since she hasn’t called me today I am confident that she has not run into such a situation. The (informal) contract that my wife and I have about calling each other allows both of us to know about each other in the absence of communication.

In a formal sense, my wife and I have models of each other’s environments and behavior. We have an informal contract to inform each other when reality deviates from the model. Thus absence of messages implies that reality matches the model.

This form of contract - no message implies your model of my situation matches my reality – is typical of all human relationships including collaborative business relationships.
Take Away Point 3: Your company is already event driven. If it weren’t it would be dead. All enterprises have to respond to events to succeed. We must recognize that the IT organization in general, and you in particular, aren’t going to make your enterprise and event-driven enterprise – it already is. The question for us in IT is whether we can help make our enterprises more agile?

Take-Away Point 4: You already have the components of EDA architecture in your enterprise stack. Furthermore, EDA and SOA are totally compatible. They have different primary objectives as we shall see later in the talk. But, in terms of architecture they are compatible: EDA can be built on top of SOA and with SOA components.

The key question for you is this: Do the incremental benefits of IT in helping your company become more agile exceed the incremental costs of building event-driven applications? I emphasize the incremental aspects of costs and benefits. Your company is already event-driven but your IT skills can help your company be more effective in responding to events. Your company already has the architectural pieces it needs to build EDA and you need only an incremental effort and cost to put it together for a different goal.

This talk is about evaluating this key question. Different people in this room will come up with different cost/benefit analyses. I hope our discussion will help each of us clarify the costs, benefits and tradeoffs for our personal enterprises and goals.
All organisms sense and respond to their environments. The stork in this picture gets streams of information from its senses – eyes, ears, touch… Likewise, the frog. When you observe storks, such as the ones in the lake outside this room, you would have noticed that they stand still doing nothing for long periods of time. The rare event – the threat or opportunity or other critical situation – is the difference between life and death. Perhaps there are 5 minutes in every hour in which a stork’s response to its environment is critical. For the remaining 55 minutes the basic functions – breathing, heart beating, digestive juices flowing – continue but the central nervous system filters out the other data streams, telling the stork that no critical event has occurred. When an opportunity – such as a meal in the form of a frog – appears, the central nervous system alerts the mechanism, and this alert is crucial. Of course, the same goes for the frog.

Animals that respond to too many false positives die. A false positive is an unreal opportunity or threat. Responding to a false positive wastes energy. An animal dies if it expends more calories hunting than it obtains by eating.

An animal that does not respond to opportunities and threats (i.e., an animal with too many false negatives) dies.

The reality of event-driven systems is that they are rarely perfect: there will be some situations that should have caused responses, and there will be responses that are inappropriate or too late. The costs of false positive and negatives, and costs of being tardy will play key roles in our analysis of the key question: Do incremental benefits exceed incremental costs?
A pride of lions hunting zebras is an example of nature’s finely-honed sense and response mechanisms. Each lion pieces together a history - a video, if you will - of the environment. The mental image includes the movement of the zebra herd, positions of predators such as hyenas, and locations of lakes and rivers. A lion gets this mental image by fusing streams of information from its senses: smell, sight, hearing,..... The lions in the herd send each other subtle signals to determine and coordinate an optimal response: which zebra to attack and how. The whole situation changes continuously and rapidly as prey and predators move.
External Awareness: Questions for you

- Does your enterprise monitor its competitors? Government agencies?
- Do people in your enterprise correlate information from multiple sources? e.g., correlate flood at a supplier's factory with deadlines for critical customers.

Here are two rhetorical questions for you. Of course, all of know what the answers are for everybody in this room. Of course, your enterprise is aware of what is going on in the world outside the enterprise. And of course, there are people in your enterprise who are putting 2 and 2 together to get 4. The point of these questions is that you already have EDA in which the components are people.
Responding to Unexpected Situations

Question for you:
- A fire has just occurred in a factory that is going to effect customers severely.
- Which of two scenarios represents your enterprise?
  4. The CEO doesn’t expect VP Mfg to say anything unless the CEO asks.
  5. The CEO expects VP Mfg to tell the CEO.

Here’s another rhetorical question. Of course, the CEO expects VPs to inform the CEO when something unusual happens. The point of this question is to emphasize the contracts that are explicit or implicit in collaborative human relationships: I have a responsibility to tell my collaborators, in a timely fashion, when something they don’t expect happens to me or my environment.
The CEO has an expectation a model of the CFO and the CFO’s environment. The contract between the CFO and CEO enables the CEO to know that the financial aspects of the company match the CEO’s current model if the CEO hasn’t heard from the CFO.

Note that the point isn’t that the absence of information implies that the situation is good or even that the situation is normal. The absence of information allows the CEO to deduce that the situation fits the model that both the CEO and CFO are currently using. The model could be that the situation is horrible, and a message from the CFO that reality deviates from the model could be positive – e.g., a line of credit has been extended.
Situational awareness, and UDOP (user-defined operational pictures) are terms widely used in the military. All components of a task force need to be aware of the global situation of a battlefield. Each user needs to have an accurate picture of the aspects – relevant to his or her role – of this operational picture. Likewise, all components of certain enterprises need to be aware of the global situation and different people and applications need to be aware of different aspects.

The picture on the right shows situational awareness in the commodity trading unit of an energy company. The company has overall goals for its Value at Risk, each location (such as the Houston office) has its own goals for VaR, and different people such as a trader in Edmonton has his or her own UDOP represented here by a “cockpit” tailored for that person.

The essence of situational awareness is that absence of communication implies that reality matches expectations (models). All the components and people just cannot keep communicating all the time.

Not all enterprises need IT support for being aware of the global situation. Many, and perhaps most, enterprises do satisfactorily with human communication and without formal organization specifically targeted at situational awareness. The question for each of us is not whether situational awareness is a good thing or not. The question is: are the benefits of IT in making our organizations more situations aware worth the added costs?
Let me repeat the four take-away points and the question for each of us.

And now, let’s move on to more technical aspects of EDA.
An event-driven application can be defined as one that executes when-then rules. When an important situation occurs, then respond appropriately.

This definition generates lots of questions. Firstly, who specifies the rules? How many types of rules? Could rules be at cross purposes? How are when-clauses evaluated? What sorts of then-clauses are used in enterprises?
A central characteristic of event-driven applications is its composition of three basic features: sense, analyze and respond. The sensing part obtains information from within and outside the enterprise. The analysis part continuously evaluates the when clauses of when-then rules by fusing information from all the sensors and determines if the then clauses should be executed. The responders execute the then clauses.

This is a depiction of the warfighter’s approach to managing threats in his/her global environment:

- He/she has sensors that detect threats – enemies, mountains, low fuel- friendly forces, air traffic control commands. The system has responders – such as alerting the pilot or initiating an automatic response without pilot mediation. And the IT system coupled with the pilot continuously analyze the streams of data arriving from sensors.
A key consequence of event-directed applications is that these applications are asynchronous. This is because the timing of events isn’t controlled by the enterprise. Certainly, the timing of events outside the enterprise – such as the time at which a competitor reduces prices significantly – are not controlled by the enterprise. Events happen when they happen.

As we discussed earlier when we talked about the informal contract between human beings and about situational awareness – the absence of communication must imply that reality matches the current model. Thus, “event-driven” implies that the contracts for some components of the system are that the components are responsible for pushing information when reality deviates from expectations. The system may poll characteristics of the environment – for example, an IT component in an enterprise may poll the web sites of its competitors periodically; but this component then pushes information to other components.

Thus a key characteristic of EDA software is that coupling between components is asynchronous.

Another view of EDA is that it is defensive programming carried to an extreme. This is because an application that couples to the external environment – including parts of the environment that may be hostile (competitors) – cannot expect the environment to follow rules laid down by the enterprise. Therefore, the coupling must, perforce, be defensive.
An airline monitoring another airlines web sites has to poll the competitor’s web site, and has to deal with unexpected changes to the site. The application must be designed to be extremely robust or it will be very very brittle. I know from sad experience working in the trenches.

In this sense we can think of EDA as taking defensive programming to the extreme.
One division of an enterprise can treat another division just as one airline treats another. In this sense, coupling within an enterprise could be extremely robust. But, we must remember that this robustness is achieved at a substantial cost. Developing applications that are based on contracts between components is much easier than when no contracts exist.
This slide is from articles by Jonathan Lurie and me in the online magazine developer.com. The articles are on sense and respond systems. Sensors are shown on the left, the event processing agents (EPAs) that analyze sensor data are shown in the middle, and the responders to the right.
The “outward-facing” components of an event-driven application are programmed extremely defensively. The sensing portion may have to poll the environment rather than have data pushed to it.
The inward-facing components are programmed less defensively. In this sense, the inward-facing components of EDA are like SOA. One can think of SOA components as being primarily inward-facing in the sense that the components have contracts specified in WSDLs.
I started out by saying that SOA and EDA are complementary and compatible. Use SOA and EDA for different goals. Let us take an extreme – a caricature – view of each to highlight the differences between the two.

The components of SOA are collaborators. SOA is an integration framework among collaborating entities. EDA may interact with components outside the enterprise – and these may not be collaborators.

In SOA the time of interaction – the time at which a service is invoked – is determined by the client. In EDA the times at which significant events occur may not be controlled by the enterprise.

In SOA service protocols and schemas are well defined. That, after all, is the goal of WS*. By contrast, in EDA, protocols and schemas may be less well defined.

In EDA absence of communication conveys information: reality fits current expectations. In SOA absence of communication conveys nothing: the server does not have a contract to tell the client when reality deviates from the client’s expectations.

I want to emphasize that the points in the slide take an extreme view of EDA and SOA to delineate the differences. They are complementary: Use SOA where synchronous interactions make sense; use EDA where asynchrony makes sense.
1. Asynchronous coupling.
   - The timing, place, and characteristics of threats and opportunities are not determined by you.
   - So sensors are responsible for pushing information; responders are not responsible for pulling it.
2. Defensive programming to the extreme.
   - Data may be unstructured and inaccurate. Protocols may be unspecified.
4. When-Then rules.

This slide is a review of the last few slides. Next, let’s spend quite some time on a possible architecture of event-driven applications.
I’m going to spend some time on this slide for two reasons. (1) To explore architectures for event-driven applications and (2) to emphasize my points that your enterprise already has many of the components of the architecture and that SOA is complementary to EDA.

I’m going to discuss the components of the architecture from left to right.

At the extreme left are the sources of data. These data sources are categorized along two axes. The first axis is the degree to which they provide well-defined interfaces, and the second axis deals with whether data is pushed by the data source or whether data must be pulled from it.

Consider the first axis: how well-defined is the interface? Web Services, databases, and applications within the enterprise (the lower blue components on the left) usually have well-defined interfaces. In some cases, these interfaces are defined using WSDL.

Data sources such as news feeds and stock ticker feeds are also (usually) well defined. The degree of meta-level description may not, however, be that of interfaces within the enterprise. For example, RSS provides a basic degree of meta-level description, and tags in blogs provide valuable contextual information; but, it’s not schematized to the same level as calling a Web Service with certain parameters and getting responses of specified types.

Hostile Web sites may have well-defined interfaces but they are under no obligation to inform your enterprise before the interfaces are changed. In some cases, the only way to get information may be through "screen-scraping" with all its difficulties and inaccuracies.

Now let’s look at the second axis: news feeds and stock-tickers may push information to the enterprise. Most of the other data sources need to be polled according to some schedule. The scheduler may need to be sophisticated. For instance, certain information – Fed setting interest rates, disclosure of gas inventories – may be made available at specific times such as Wednesdays at 1PM Central Time. These sites may need to be polled frequently just before the time at which the information is usually made available and then not polled at all. The scheduler is important enough to deserve its own explicit component in the architecture chart.

The orange components that interface with the data sources are sensors – they poll or accept data from external sources and convert them to standard schemas.

Events from the sensors are passed to the Enterprise Service Bus. The ESB may carry out further translation of the schemas. The next several steps in processing are carried out by Event Processing Agents (EPAs) shown in green. Often an event passed to the ESB by a sensor, say a news source, may be processed by a sequence of EPAs – for instance first by an EPA that analyzes natural language text and then by a Time Series EPA that carries out analyses on the history of results generated by the text analyzer.

Responses to significant events are either alerts to people (shown in this picture by the alert engine on the top right) or to databases and internal applications (shown in the bottom left) or to external partners such as electronic markets.
This slide repeats the take-away points: these points are even more relevant after studying an event-directed architecture.

Next let’s look at the down-sides, the pitfalls, of developing an event-driven application. It’s best to think through the costs and benefits carefully before embarking on development of EDA.
An EDA system is only as efficacious as the appropriateness of when-then rules for each user. The efficacy depends on whether rules are tailored for each user’s specific role and individual needs.

Developing and tuning rules takes time and thought. A major cost is the time required from the end user – the business user – to develop rules truly useful to that end user. We will explore this cost later in the presentation.

Errors: alerts to non-critical events and non-responses to critical events are possible.

Why are these problems inevitable? Let’s run through problems and how you can ameliorate the difficulties.
Errors are inevitable for several reasons. Rules may be imperfectly specified. This is because tuning and re-tuning a rule to match a user’s specific changing needs takes the user’s time.

The system may not evaluate when-clauses accurately. Consider a when-clause that fuses data from multiple event streams – for example, a when-clause that is true when the number of news stories in the last 15 minutes about changes in the senior management of a company coupled with a significant change in the price of that company’s stock. Each of the components of the when-clause may be evaluated imperfectly because this kind of match is fuzzy.

We may be tempted to reduce false negatives by casting a very wide net. But that increases the rate of generation of false positives. Likewise, narrowing the when-clauses to reduce the frequency of false positives may increase the rate of false negatives.
Most responses alert people.

- Perception: false positives cost much less than false negatives.
- Too many false positives: “so, turn that thing off.”

Correct evaluation should be based on total costs of all the false positives and negatives over time.

In most of the event-driven applications that I’ve seen, responses are mediated through human beings. Today, there are few applications that automatically invoke a mechanical response – say automatically carry out a trade on an arbitrage opportunity. Because human beings mediate responses, we tend to think that the cost of a false positive is low…. After all, what is the cost of sending an alert to a person? The person looks at the alert, perhaps analyzes it for a short while, and then discards it. If however, too many false positives are generated, the system will be ignored or turned off.
Inevitable Down-Sides of EDA: No Rules

- People are busy.
- Learning tools for specifying rules takes time.
- So, rules aren’t specified, or are specified in a cursory fashion.
- Destroys business value of EDA.

Why don’t business users specify rules when (we know that) doing so will make them more efficient?

Tuning and re-tuning rules takes effort. Learning what rules can be specified – what sorts of event-processing agents exist (text analysis, time series analysis, video analysis,…) – takes time. People are busy and can’t spare this time.

The absence of good when-then rules destroys the business proposition of event-driven architectures.
Let’s explore some ways of overcoming the hurdle of requiring up-front effort from the end user.

A tempting idea is to have IT or some other central organization develop rules for the business user. That approach doesn’t work. It doesn’t work because IT doesn’t know precisely the events to which the business user wants to respond or the most appropriate responses. Central organizations may cast too wide a net in specifying when-clauses because they value the end user’s time less than the end user does, and as a consequence the cost of a false positive (the end-user’s time) for the central organization is less than the cost of a false positive to the end user. This may result in the system being ignored by the end user.

A good strategy is to work with one or two business users to develop a library of rule templates. Individual business users can then specify rules merely by specifying parameters in the templates. The generation of templates requires energy, intelligence and imagination. Tailoring a template to an individual’s specific needs by specifying parameters is relatively easy. The problem is getting the time from business users in developing a library of rule templates.

Once a library of templates has been developed, the library should be stored in a repository that can be searched by roles. For instance, a purchasing manager can search the template library for templates that have been helpful to other purchasing managers.

One approach is to have the system “learn” when-clauses from examples provided by end users. This approach is used in categorizing text: users provide examples of text that fall into a category – e.g., deals with changes in senior management of companies – and examples of text that don’t fall into the category. A learning algorithm can determine how to categorize new text samples (though not without error). Some business users prefer to have the system learn the rules, but even this requires the business user’s time. Example suites have to be provided and the resulting rules have to be tuned. Learning systems do not take away the need for the business user’s time.

Think through the cost of developing rules. It’s a significant part of the overall costs.
Is Anything Missing in your Software Stack?

- Event Process Agents
  2. Machine can “learn” the critical condition from positive and negative examples
  3. Users can specify critical conditions
     - SQL-like queries
     - Fuzzy matches
     - Statistical operators
     - Regular expressions
     - CEP

Now let’s look at another aspect of the incremental cost: the development of components needed for EDA and the effort in integrating these components to provide EDA functionality.

The components that are most likely to be missing are the “cognition” components – the EPAs. You should think through what kinds of EPAs are appropriate for your application. Start with one type, start with one or two rules, show value and then keep adding capability.
Estimating Performance Requirements

- Delay from occurrence of condition to initiation of response: Minutes? Sub-seconds?
- Number of data sources: Tens, Hundreds?
- Numbers of rule templates: Tens, Hundreds?
- Numbers of users?
- Numbers of rules?

Observation: Many enterprises overshoot: they estimate greater performance requirements than they need.

In my experience, many enterprises over-estimate their EDA performance requirements. Why?

All enterprises generate events rapidly. Consider telephone calls to a customer service center. If we treat each call initiation and termination as events, the service center itself will generate huge rates of events. If we take an inventory of all the events in an enterprise rates of 10,000 to 100,000 per second are not unusual. But these are rates are not relevant for event-driven applications.

Start with identifying the situations that need responses. Most of the events in an enterprise are “normal” or “usual” and will not identify a response-requiring situation. Usual events can be filtered out at the source. You may hear terms such as edge-filtering or filtering at the periphery. Events that are filtered out at the edge can be stored at the edge, and used later if necessary. The rate of generation of events that are likely to identify response-requiring situations is usually relatively small.

If the response is mediated by a human being, then a response delay in the order of a few seconds or possibly even minutes is acceptable. Sub-second delays don’t help people.

In many situations, an event-driven application can be partitioned into systems for responding to different kinds of situations. Thus though the total event rate may be high, the rate for each EDA may be much smaller.

The kind of architecture you use will depend a great deal on whether you need sub-second response or minute response, on whether you have events at a hundred per second or several thousand per second, on the number of data sources, and on the types of when-clauses. A central component for many applications may be your enterprise database provided responses are mediated by people and relevant events are generated in tens or hundreds per second.

The ideal situation to demonstrate business value is when you can identify a handful of rule templates that can be tailored by large numbers of users to produce huge numbers (many thousands) of individual rules, where the when-clauses are parametric and SQL-like (as opposed to analyzing text or video).

Think through performance requirements very carefully. Remember that your enterprise is already event-driven. Your goal is to improve its event-driven capability. Chances are that for incremental improvement you don’t need an IT system that processes many thousands of events per second and detects significant situations a few milliseconds after they occur.
As I said in the previous slide, an EDA system can be designed for the common case shown on this slide. For this common case, your existing enterprise software components – databases, rules engines, ESBs – are adequate.
All enterprises are event driven. So, why is EDA uptake slow? Indeed, *is EDA uptake slow?*

Simple event processing – filtering out a single event stream – is widely used. And a few (2 – 5) event streams are correlated in some applications. But, these applications aren’t usually thought of within the overall rubric of event-driven apps. They are thought of as unique designs for unique requirements rather than as instances of systematic design for a large class of apps.
I’ve noticed that EDA uptake is relatively rapid in some areas such as financial trading, intrusion detection, IT infrastructure management and defense. Why these spaces and not others?

Three reasons. (1) Perhaps most importantly, in these spaces there are well-defined small groups or lines of business responsible for responding to critical events. Traders are responsible for ensuring Values at Risk do not exceed thresholds. IT security managers are responsible for preventing intrusion into their networks. These small groups see the benefits of event-driven applications because it effects their daily work. Yes, it is true that all companies are event-driven but if the responsibilities for responding to critical events are diffuse and spread all over the company, there is no single line of business that will take the trouble of developing and maintaining an event-driven application.

(2) The business proposition in these spaces is very clear. The business proposition in other spaces is less clear, and existing human-intensive solutions may be adequate.

(3) Performance in these vertical areas is important. Thus the value from better performance of an IT solution is evident.
Should you build EDA?

- Your company is already event driven; you have many of the software components. Question:
  Do incremental benefits exceed incremental costs?

Should you build an event-driven application? The answer depends on you and your company. We keep coming back to the key question dealing with incremental benefits and costs.

Spend a lot of time thinking through this question. Let’s explore this question in the next few slides.
Should you build EDA? Do you have:

1. Applications:
   - Apps that sense and respond to the environment and that benefit from automation?
   - Evolving middleware that can benefit from asynchronous coupling?
2. Responsibility in a single group or LOB?
3. Performance requirements not met?
4. Small numbers of data sources and rule templates satisfying large numbers of users?

(1) Think through the business application. What business processes already exist in your enterprise that respond to critical situations? Which of these can benefit from automation?

(2) Does the responsibility for responding to some critical situations reside within a clearly-identified group? If it does, you can demonstrate business value much more easily.

(3) Are performance requirements being met today with the existing approach? How much better can performance be with IT support? Is this incremental improvement in performance valuable to the business activity of responding to critical conditions?

(4) You will find that developing an app is easier if the number of data sources is small, and there are a small number of rule templates that can be tailored and used by a large number of users.

Keep in mind that a large part of the costs is the time required from business users. Yes, eventually this time pays off. But, getting the business user’s time can be a hurdle.
Should you build EDA?

- Are costs of false positives and negatives smaller than benefits of EDA?
- Do you have a senior manager in a LOB who will make all the hard business work happen?

(Building effective EDA is 95% effort by business people and only 5% effort in technology.)

Keep in mind that some false positives and negatives are likely. Emphasize the incremental business value of EDA that is provided even though there are a few errors. Admit up front that an event-driven app may not be infallible.

A lot of the cost is the time required from business users. Ensure that somebody can help overcome the initial hurdle of providing this time. Once you demonstrate value with a few rules, you may get business users to volunteer their time. You may need help from a senior manager when you are starting out.

Building an effective event-driven application is mostly effort from business people. Certainly building a sense-and-respond enterprise or an agile enterprise is very much a business issue rather than an IT issue. Have a look at Steve Haeckel’s book on the Sense and Respond Enterprise.
I’d be happy to talk to you next time I’m invited about the sequence of steps you go through to build an event-driven app. And, what I’ve learned from doing this in the trenches… the gotchas that got me and shouldn’t get you.

But, this is for another time.
Here again are the take-away points, and the key question for you.

**Take-Away Points and THANKS!**

1. EDA: Global situational awareness
2. EDA: Unit “knows” reality matches model until the unit gets a message.
   - Your enterprise:
   5. Is already event-directed.
   6. Has the software components including SOA
   - Your Question: *incremental* costs vs. benefits?
Getting Business Users to Specify Rules is Hard

- Different roles have different sets of rules. Who specifies the rules?
- How many rule templates? Tens, hundreds?
- How many rules? Hundreds, thousands?
- How are rules specified? Language, visual UI, positive and negative examples? Fill in templates?
- Who can turn a rule off?
- What if rules are at cross-purposes?