

# Sense and Respond Systems

K. Mani Chandy  
Computer Science  
California Institute of Technology  
mani@cs.caltech.edu

*Sense and respond systems sense, and then respond, to opportunities and threats. Sense and respond applications arise in a variety of settings including homeland security, finance, IT infrastructure management and personal productivity tools. This talk surveys sense and respond systems, identifies application spaces and discusses key technologies.*

## **INTRODUCTION**

Sense and respond systems sense what is going on in their environments and respond appropriately. Groups of living things - such as ant colonies and human institutions - are sense and respond systems because they sense and respond to collective opportunities and threats. Sense and respond IT applications amplify an enterprise's capability to sense and respond to critical conditions.

The term sense and respond systems has been widely used for some time; see for instance: [Haeckel93, Haeckel99]. A review of government, private and NGO (Non-Governmental Organizational) responses to natural and man-made disasters shows that success in responding to a changing environment depends much more on leadership, organization, and training than on information technology (IT). This paper, however, restricts attention to IT issues.

Sense and respond applications process events. Systems that process events have been studied extensively [Luckham, Ranade, Chandy, Schulte]. This paper is based on this earlier work.

## **When-Then Rules**

The essential characteristic of a sense and respond system is that it executes a set of *when-then* rules of the form *when* a condition holds *then* respond in the prescribed way. The designs of sense and respond applications vary widely depending on the types of *when* and *then* clauses. No single architecture is appropriate for the entire range of *when* and *then* clauses. The space of designs is discussed later in the paper.

**Separating concerns about when-clauses and then-clauses:** The design of sense and respond (S&R)

applications is simplified by separating two concerns: the detection of the critical condition (evaluation of the *when*-clause) and execution of the response (the *then*-clause). For our purposes, the execution of the *then*-clause can be reduced to sending a message: This message may be an invocation of a complex orchestration of business processes. Business process management (BPM) has been studied extensively, and many BPM tools exist; so no new software tools, specific to S&R, are required to orchestrate business processes. Separating concerns about *when* and *then* clauses enables us to restrict *then*-clauses to sending a message where the message is usually sent to one of the following:

1. Another rule (see the paragraph on rule composition below)
2. A human being – this is an alert in the form of a phone call, instant message, email, or some other medium.
3. A business process which may, in turn invoke a complex orchestration of business processes including updates to databases and Web sites.

**Composition of rules:** Simpler *when-then* rules may be composed to form more complex *when-then* rules: the *then* clause of a rule may generate events (new data items) that are inspected by the *when* clause of another rule, causing that rule to execute.

**Incremental computation of then-clauses:** In some applications a *then*-clause may be executed repeatedly with small changes to the data set. For example, consider path-planning for a vehicle. The optimal path is computed initially with a given set of data. The *when*-clause detects a change in this data set; for example, the *when*-clause detects a new obstacle in the optimal path. The *then*-clause recomputes the optimal path with this change in the data; the change is only to a very small fraction of the data, and hence the recomputation can be very rapid. Incremental algorithms for

recomputing *then*-clauses are an important area of research and are discussed, very briefly, later in a subsection entitled high-performance complex event processing.

### Evaluating Sense and Respond Applications

A simplistic, but nevertheless useful, way of evaluating sense and respond systems is in terms of the following cost – benefit analysis. The benefits are obtained from appropriate responses to critical conditions. The costs include the following three components:

1. The costs of false positives.
2. The costs of false negatives
3. The incremental costs of implementing and running a sense and respond system as opposed to a more conventional system.

The cost of a false positive is the cost of executing a response to a condition that did not arise. It is the cost of executing the *then*-clause of a *when-then* rule while the *when*-clause is false. For example, it is the cost of ordering the evacuation of a city when a hurricane hits landfall elsewhere or is milder than expected.

By symmetry, the cost of a false negative is the cost of not executing the *then*-clause of a *when-then* rule though the *when*-clause is true. It is the cost of a hurricane hitting a city that should have been evacuated.

Evaluating sense and respond applications in terms of false positives and false negatives is simplistic because sensing and responding are not binary activities that can be compartmentalized into positive and negative bins. Some responses may be of the right form but too tardy to have maximum benefit. Other responses may be timely but inappropriate to varying degrees. Though evaluation in terms of false positives and negatives is simplistic, we shall use it in this paper because it provides a useful framework in designing systems.

Feedback from the environment is used in some sense and respond applications. For example, a *when-then* rule used in email handlers is *when* a spam message is received *then* place the message in a spam folder. Users provide feedback to the system telling it which messages that the system identified as non-spam are in fact spam, and which messages identified as spam are non-spam. The feedback should reduce the costs of false positives and negatives over time. The process of incorporating this feedback can also be implemented in terms of *when-then* rules: *when* the feedback is provided *then* modify parameters in some other rule.

### THE ENVIRONMENT

Opportunities and threats to an organization arise from

both within and outside the organization. Competitors and markets can generate events that require responses. S&R (Sense and Respond) applications must, perforce, monitor conditions outside the organization.

Events generated outside the organization are not within the organization's control. Competitors do not structure data in schemas that suit the enterprise, nor do they deliver competitive data to an enterprise using its Web services. The externality of critical events has profound implications for the design of S&R systems, some of which are listed next.

1. **Asynchrony:** Events are not generated at times chosen by the enterprise. Events may happen unexpectedly, and they may occur in bursts.
2. **Data delivery:** Data about critical external events will not be delivered to the enterprise; the enterprise has to search for this information.
3. **Data accuracy:** The data obtained may be inaccurate and unstructured.
4. **Data schemas:** Data about external events may be structured in well-defined schemas, unstructured in the form of natural language text or images, or partially structured in tables on Web pages. The schemas often change.
5. **Delays:** Because an enterprise has to search for external data, a substantial delay may occur between the occurrence of a critical event and its capture by the enterprise.

The impact of these factors on the design of S&R applications will be discussed later in this paper. The central issue due to the external environment is the degree of coupling between components: connections to competitors or other organizations must, perforce, be "loose" in the sense that the enterprise cannot depend on the organization from which it is obtaining data.

An enterprise may use S&R applications for collaboration between units *within* the enterprise itself. In this case, some of difficulties of dealing with an external environment can be ameliorated – for instance units within an enterprise may not be permitted to change data schemas without informing other units within the enterprise. S&R systems used exclusively within an enterprise are, in effect, EAI (Enterprise Application Integration) systems where the degree of coupling between units is usually "tight" because components can depend on other components to behave according to pre-specified contracts.

Loose coupling implies less dependence on other components; tight coupling implies greater dependence. Measures of the looseness of coupling include the five listed earlier: asynchrony, data delivery, data accuracy, data schemas and delays.

Even within an enterprise, architects may choose to use

loose coupling in which each unit of the enterprise treats other units as undependable. This makes for more complex programs but results in greater resilience. The trend towards looser coupling is evidenced in the increasing popularity of SOA (Service Oriented Architecture). A next step in this evolution is a possible transition of a few applications from SOA to EDA (Event Driven Architecture). This transition has its costs: defensive programming – in which each component depends less on other components – requires more careful design. The limiting case of defensive programming is where each component implements its own S&R application.

## **TYPES OF SENSE AND RESPOND APPLICATIONS**

The design of S&R applications depends a great deal on the type of application. There are many important niches of S&R applications. Some of these niches are far apart in the design space, and so a single S&R platform is not appropriate for all niches. Next we explore characteristics of S&R applications, and then discuss niche applications in terms of constellations of these characteristics. These characteristics are based on different features of *when* and *then* clauses.

### **Characteristics of S&R Applications**

The characteristics of S&R applications can be studied by analyzing the following 10 questions. This list of 10 questions is not the ultimate list, but it does help to think about the space of applications.

**1. Is the response an alert to a person or an automatic machine action?** The mission of many S&R applications is to alert a human about a potential threat or opportunity; a person makes the judgment about whether the potential critical condition is truly critical. An S & R application may alert officials in Homeland Security about a potential terrorist threat, but these officials evaluate intelligence to make the final call on whether to raise the state of preparedness. The *then*-clauses in many other applications invoke software or mechanical processes automatically. For instance *when* the levels of carbon monoxide in a building are elevated *then* turn on sprinklers in the building automatically. Whether the response is an alert to a person or an automatic machine action has two consequences. Millisecond differences in response times are insignificant when the mission is to alert a person because human reaction times are usually a few seconds. Secondly, the cost of a false positive, when the mission is to alert a person is the cost of a false alert: the alerted person has to evaluate the alert and then discard it. The cost of a false positive when the response is by a machine may be very high: turning on sprinklers in buildings, when there is

no fire, is an example.

- 2. Responses in minutes or in sub-seconds?** Responses in minutes or even hours are adequate for many, and perhaps most, applications. Some applications, particularly in the management of IT infrastructure and the military, require responses in sub-seconds. The required latency of response impacts the design; for instance, S&R applications with responses in minutes can often be designed by composing conventional tools such as databases or rules engines. S&R applications with sub-second responses may require in-memory data and sophisticated incremental algorithms on streams.
- 3. Are rates at which events are generated in tens of thousands per second, or a few per second?** Some applications, such as S&R for stock trading, have *when* clauses that require the analysis of events generated at several thousand per second. In a large number of enterprise applications, however, each rule requires the analysis of only a few events per second. High volumes with rapid response require parallel execution.
- 4. When-clauses based on single event or history of events?** Many applications require each individual event to be analyzed and responded to, independent of previous events or other event streams. For example, messages of certain types – such as order cancellation - from important customers may require immediate response. A few applications require analysis of the history of events: for example to determine whether the 13-week moving point average of a stock price crosses its 52-week moving point average, the history of stock prices must be evaluated. Applications that make decisions on each event independently of other events are simpler and need to store less information than applications that make decisions based on histories.
- 5. Unstructured or structured data or both?** In many applications, the external data that must be analyzed is unstructured: news stories, filings in government agencies, blogs and emails. The data in some applications conform to well-defined schema. Analysis of unstructured data – text, images or video – requires different technologies than extraction of information from structured data.
- 6. Unstructured or structured queries?** Queries in many applications are unstructured. Consider, for instance, S&R systems used after a disaster such as a hurricane. People looking for pets may have queries (or, equivalently, predicates that they wish to sense) of the form: *When* a cat, colored black, with white paws, about 10 pounds, is found within 5 miles of Magnolia Street *then* call the following number.

The types of *when* clauses can be extremely varied, especially in applications responding to disasters such as hurricanes and earthquakes. Queries in some applications are structured in well-defined schemas; for example: *When* the price of product **X** in electronic market **Y** is **Z** % below its internal company price *then* send an instant message to the purchasing manager of the product. Many specific queries may be generated from this schema by specifying the parameters, **X**, **Y** and **Z**. Applications in which queries are unstructured have *then*-clauses that are (typically) alerts to people – as opposed to automatic machine responses – because the frequency of false positives is very high. The technologies used in analyzing unstructured queries are different from those – such as database and XML tools – used in dealing with structured queries.

**7. Evaluate a single stream or fuse multiple streams?** There are many applications in which the *when*-clause of a *when-then* rule is a condition on a single event stream. For example, a rule that alerts a trader to a stock trading opportunity may evaluate conditions based on the single stream of stock prices. Some applications require correlation or fusing of information in multiple streams. For example, a trader may want to be alerted about any company when there are news stories about changes in the management of that company **and** the company's stock price changes significantly from the previous day. One measure of the complexity of an application is the number of streams that must be correlated in typical *when* clauses. An S&R application may operate on hundreds of streams, but each individual *when-then* rule (in many applications) operates on only a small number of streams.

**8. How are parameters estimated in the absence of messages?** When the last message about a parameter – say the price of a stock – arrived some time in the past, the S&R application has to *estimate* the value of the parameter at the current time; this is because the *when*-clause is a predicate on state histories up to the current time. An assumption made by many applications is that the value of a parameter changes only when a message about the new value of the parameter is received. For example, if the stock price of a company is \$200 at 10 AM, and the next message received about this stock price is that it is \$201 at 10:01 AM, the application may treat the stock price as unchanged at \$200 from 10 AM till just before 10:01 AM. Consider, by contrast, an application that is required to interdict terrorist vehicles; the application must estimate the location and trajectory of the vehicle even when it hasn't received messages. The design of applications is simpler if we can assume that parameter values change only when messages are

received: for example, if a database is updated with the most recent values of all parameters, then the database contains a description of the state of the system at the current time. This allows us to execute database queries only when the state changes, i.e., when the database is updated, or when time moves forward by a specified amount. By contrast, the design of applications in which parameter values – such as locations of vehicles – must be estimated continuously, is more complex.

**9. Events from collaborators or competitors?** As discussed earlier, S&R used entirely within an enterprise is an example of EAI: here problems of dealing with uncertain environments can be ameliorated by using appropriate protocols among collaborating units. Events generated by competitors may have different schemas and different meanings at different points in time.

**10. What are the costs of false positives and false negatives?** Many applications have relatively low costs of false positives; some have only moderate costs of false negatives. A few applications – determining to evacuate a city in the path of a hurricane – have enormous costs of both false positives and false negatives. The designs of S&R systems depend on these costs.

## THE SPACE OF APPLICATIONS

Different types of S&R applications require different designs. Since no single platform is appropriate for all S&R applications, architects of S&R applications should think carefully about the architecture most appropriate for their application. One way to characterize their application is in terms of the issues raised in the previous section.

The architectures of an S&R platform may be changed several times because the answers to these questions have not been thought through. The platforms may have to be reprogrammed each time the architecture changes; the reprogramming will be due not to the usual culprit (creeping *incremental* functionality) but because of creeping *changing* functionality. If the answers given by business drivers to the above questions change then the architecture of the application will change as well. An apparently easy way to answer a question about choice between alternatives is to make no choice: Make the architecture work well for all alternatives. This approach is not feasible for the design of S&R applications.

Next, we discuss niches in the design space, with different architectures appropriate for different niches.

### **Simple Event Processing (SEP)**

Many applications require only simple event processing:

in SEP the choices for all 10 questions are the simpler choices. Responses are alerts to human beings. Response times of several minutes (and in many cases even a few hours) are adequate. Rates of event generation are in hundreds per second rather than tens of thousands per second. Event data and queries are structured in well-defined schemas. Events are generated by collaborators rather than competitors. Messages are received each time relevant parameters change. *When*-clauses are conditions on single events or are very simple conditions – such as moving point averages – on time windows.

An example of an application in this niche of the design space comes from the compliance area. An enterprise wishes to carry out plan-versus-actual comparisons, alerting appropriate officers when significant deviations occur. An official may choose to escalate the alert up the management hierarchy. The application monitors events dealing with income and expenditures, compares plans with actual data, generates alerts that may include a pointer to a location in a data cube of a business intelligence tool. A person getting an alert, starts analyzing the potential deviation in actual data from the plan by exploring the data cube at the point suggested in the alert. If this exploration suggests that the alert is significant, the alert is escalated up the hierarchy.

This application is a productivity-improvement application: people in the enterprise don't have to scan data continuously – they are alerted when their attention is required. The process of escalation up the management hierarchy reduces the frequency of false positives higher in the hierarchy. Productivity improves because the application is an attention-concentrator.

The architecture for this application is centered on a database. Events within the enterprise are fed to message queues or an enterprise service bus (ESB) which routes appropriate events to that database. The database is updated with information from relevant events. *When*-clauses are implemented as persistent database queries structured in SQL or XQuery notations.

Even if the database does not support persistent queries, the following straightforward solution is adequate: Execute queries periodically to determine if new alerts should be generated. The *then*-clause generates an email or instant message that may include a link to a data cube. Workflow tools are employed to escalate alerts. The components of this architecture – databases, message queues or ESBs, and workflow – are found in standard enterprise software stacks.

Many S&R applications in enterprises are SEP applications. These applications don't receive much attention in the literature because the components of the architecture are drawn from the standard software stack.

## Simple Event Processing (SEP) with External Data

Consider an application to improve the productivity of purchasing managers. The application polls the Web sites of multiple electronic marketplaces to obtain the prices of items. The application generates an alert to the appropriate purchasing manager if the price in a marketplace is significantly lower than the price paid by the enterprise. The sole difference between this example and the example of the previous section is that data is obtained from outside the enterprise. In this case, the external sources of data – such as electronic marketplaces – may be collaborators that have Web sites that can be accessed through Web services.

The architecture for this application is the same as that given in the previous section for SEP except that there is one addition: a connector to external data sources. The connector is implemented as a process that polls external data sources and extracts relevant information. This information is extracted from structured data (e.g., with XML schemas) using tools such as XQuery, and from unstructured data (HTML or untagged data) using "screen-scraping" techniques.

Libraries of connectors to external data sources – though perhaps not be part of the standard enterprise software stack. – are available from companies that offer EAI tools. The architecture for these applications is the architecture for SEP with the addition of external connectors.

The SEP architecture with external connectors is also the architecture used in applications that consolidate structured information from multiple data sources, discover when critical conditions hold, and then alert people. An example is an application that discovers when the price of an item (such as a book with a specified ISBN number) offered by several vendors on the Web, is below a threshold specified by the user.

## SEP with Unstructured *When*-Clauses

*When*-clauses in several applications are unstructured. In many of these applications response times in minutes are acceptable. For example after a disaster, such as a flood, people need to find other people, their pets and their belongings. Designers cannot foresee every possible data structure and create appropriate database tables in advance. A *when*-clause identifying a pet cat or a framed photograph may have to be in natural language text because no schemas were pre-specified for these objects.

Responses from such S&R applications may be messages identifying people with information about the lost objects. These messages can be sent to cell phones in hours after the query is created, and yet be useful. S&R applications with relatively high rates of false

positives and negatives are acceptable because they are better than no application at all.

The central component of the architecture for this type of application is a search engine. A user interface is provided through a Web site or portal. Web sites and search engines have been used by people in recent disasters. A simple, and extremely valuable, feature missing from many existing free services is asynchrony: the feature that allows a user to initiate a search which proceeds asynchronously in the background and then alerts the user by calling a cell phone. Many evacuees have to share a few computers in a shelter, such as a school or a sports stadium, but many evacuees may have their own cell phones with them.

### SEP with High Performance

Several applications are similar in structure to the SEP examples given earlier except that performance requirements are stringent: the rates at which events are generated are high (thousands per second and more) and response times are required to be small – a few seconds at most. Apart from performance requirements, these applications have the characteristics of SEP. These characteristics include structured data and queries; data sources from collaborators and not competitors: changes in data are communicated directly to the application – so complex algorithms for the estimation of parameters are not required; and often *when*-clauses can be structured as SQL queries. Some stock-market trading applications have these features.

An architecture for these applications is identical to the architecture, given earlier, for SEP with moderate performance requirements except for one difference: The database must be an in-memory database to deal with the stringent performance requirements.

### Complex When-Clauses

Next we turn attention to **complex event processing** (CEP). in which the *when*-clauses specify complex patterns that cannot be expressed conveniently in SQL or XQuery or similar notations. For example patterns may be specified by complex regular expressions over the history of multiple streams. The times at which events occur can be a critical aspect of the pattern: for instance, in a logistics application, an anomaly is signaled by a package that is shipped at one transshipment point and that does not arrive at the next point within a specified time. An RFID application for luggage handling at airports has patterns that indicate interference between multiple RFID tags. For instance an RFID reader that is reading tags of baggage being loaded on a conveyor belt into an airplane may get spurious readings when a tug goes by carrying baggage

to an adjacent plane. A *when*-clause may specify the pattern may that identifies this type of event. This type of *when*-clause cannot be expressed conveniently in SQL or XQuery.

### Low-Performance CEP

Some applications may have complex *when*-clauses but only modest performance requirements: event rates may be low or acceptable response times may be long. Other applications may have complex *when*-clauses and stringent performance requirements. Therefore, we distinguish low-performance CEP from high-performance CEP.

An extreme situation of a low-performance CEP application is a business intelligence (BI) application in which the *when-then* rule is: *when anything anomalous* is detected *then* alert the appropriate organization. The anomaly is determined by BI experts possibly over several days.

The central components of the architecture for low-performance CEP S&R applications are data mining and BI tools. Often, data is batched and then analyzed overnight, over a weekend, or over longer periods. Continuous recomputation, when each new event arrives, is not required.

### High-Performance CEP

Standard enterprise software stacks do not contain the central components of the architecture for high-performance CEP. A critical problem in high-performance CEP is incremental computation of complex *when*-clauses: the rapid re-evaluation of the *when* and *then* clauses with the arrival of each new event. Ideally, the computational time required to incorporate each new event should be independent of the length of the event history: for instance the time to compute statistic, such as a linear regression, over a moving window should be independent of the size of the window [Capponi05]. Algorithms in which the computational complexity increases slowly (low-degree polynomial) with the window size are acceptable in many applications. The development of these components requires research in incremental algorithms for signal processing, statistics, change detection, state estimation and related fields.

The incremental algorithms required for high-performance CEP vary widely from one application to the next. Algorithms for rapid computation of orthogonal regression with each new event may be critical for some applications and irrelevant in others. Here too, identifying the application space is critical.

## Summary of the Application Space

The analysis of the application space suggests a few central issues. The architectures for different applications vary widely. Think through the application space very carefully before embarking on the development of an S&R application. Many S&R applications require only low-performance SEP; a few require high-performance SEP. The components of the architectures for these applications are found in standard enterprise software stacks. Some applications require low-performance CEP, and a few require high-performance CEP. Components for high-performance CEP are not part of the standard enterprise software stack. Here too, algorithms are very different from application to application. Therefore a careful characterization of the application space is critical.

## COMPUTATIONAL MODELS AND THEORY

Next, we merely point to some of the theoretical work in the area of S&R system design.

Compositions of multiple *when-then* rules can be represented by directed graphs in which a node represents one of the following:

1. A *when-then* rule.
2. A *data source* such as a stock ticker that pushes stock price information to the application or a process that pulls information by polling a Web site.
3. A *data sink* representing a message sent in a *then*-clause; this message invokes a business process or alerts a person.

A directed edge from node  $v$  to node  $w$  represents data generated by node  $v$  and read by node  $w$  [Manohar04, Zimmerman05]. Usually, the graph is acyclic with data sources represented by nodes with no input edges, and data sinks represented by nodes with no output edges.

A source node is activated when data arrives from the environment. An item of data can be represented by a "token" on an edge of the graph as in Petri nets or dataflow graphs. Rule nodes and sink nodes are activated when appropriate data items (equivalently, appropriate tokens) arrive at *all* the node's input edges. What does "appropriate data items arriving on all the input edges" mean? The answer depends on the application and this answer impacts the design of the application in a fundamental way: see question number 8 in the list of 10 questions that characterize S&R applications.

Data arriving from a source node may cause several rule activations; the activation of a rule may cause it to generate new events which, in turn, cause activations of further rules.

## References

[Haeckel93] S. Haeckel and N. Rolan, "Managing by Wire," Harvard Business Review. September 1, (1993).

[Haeckel99] S. Haeckel, "The Adaptive Enterprise: Creating and Leading Sense and Respond Applications," Harvard Business School Press. (1999).

[Ranade]: "The Power of Now: How Winning Companies Sense and Respond to Change Using Real-Time Technology", McGraw-Hill 1999.

[Luckham] "The Power of Events: An Introduction to Complex Event Processing in Distributed Event Systems," Addison-Wesley, 2002

[Schulte] R. Schulte "Using Events for Business Benefit", Business Integration Journal, May 2004

[Manohar] R. Manohar and K. M. Chandy: "Delta dataflow Networks for Event Stream Processing", Proceedings IASTED Conference on Parallel and Distributed Computing and Systems, November 2004

[Zimmerman] D. Zimmerman and K. M. Chandy "A Parallel Algorithm for Correlating Event Streams" Proceedings IEEE Parallel and Distributed Programming Symposium, April 2005

[Chandy] K. M. Chandy and J. Lurie Carmona, "The Event Web Series": Developer.com, 2005

[Capponi] A. Capponi, K., M. Chandy, I. Fatkullen: "Predicate Signaling", California Institute of Technology, CS Tech Report 2005.