

# Rapid and Coordinated Responses: Patterns for Real-Time QoS Adaptation

Paul Rubel, Joseph Loyall, Michael Atighetchi, Partha Pal, Richard Schantz  
BBN Technologies

## Abstract

*Many Distributed Real-time Embedded (DRE) applications need to adapt to changing conditions. They are often subject to dynamic conditions, such as resource contention and availability. These conditions are more dynamic in DRE systems than in traditional static real-time systems because DRE systems are spread among networked, often heterogeneous platforms, which can consist of a variable number of communicating nodes, which use resources (such as wireless, tactical, and satellite networks) that are subject to noise, failure, or attack. Adaptation in these systems has to satisfy two seemingly conflicting forces. The first is that adaptation has to fit within the real-time constraints of the DRE system and must rely on whatever information is available at that time. Taking too much time to adapt can lead to missed deadlines and/or failure to meet mission critical requirements. The second force is that adaptations must be consistent with the requirements of the entire distributed system, i.e., local adaptations must work in conjunction with other local adaptations and be consistent with the requirements of the entire DRE system or system of systems. This paper presents two constituent patterns and a compound pattern for implementing rapid local adaptations, reacting and controlling limited local conditions, and coordinating multiple local adaptations for subsystem or system-wide requirements. These patterns are necessary for DRE systems that can maintain their critical QoS and mission requirements in dynamic environments.*

## Response Pattern Template

---

The patterns presented below, *Rapid Local Response*, *Coordinated Response*, and *Local and Coordinated Response* show how to respond in adaptive DRE systems. Due to their commonality, presenting each separately would entail quite a bit of overlap. The following common information template is used to factor out commonality between the patterns.

---

**Context.** An application where decisions on operating modes or responses must be made.

**Problem.** The “OODA loop” (Observation, Orientation, Decision, and Action)[1] is a generic model for making decisions. The following patterns describe how to implement decision making in different contexts: when responses need to be carried out quickly, when coordinated response is called for, and when a combination of rapid and coordinated responses is called for.

Solving this problem requires the resolution of the following forces that are common to all the patterns:

- Responses need to be carried out and designed in a systematic manner

**Solution.** Create a system that can assess the current situation and take appropriate action.

In Detail: Define a set of *sensors* that can be used to measure the system state. Define a set of *actuators* that can respond to events. Tie the two together using *contracts*. When the contracts, using sensor data, see that a response is required, they will run an actuator.

**Structure.** The Reaction Patterns have the following three participants, shown in Figure 1:

- *Sensors* provide the observe functionality and gather information about the system. They feed this information to contracts.
- *Contracts* provide orientation and decision. They gather and aggregate information and make decisions about the best way for the system to act. The Contract Pattern[2] provides guidance on structuring contracts that can be used to provide the management or control this role requires.

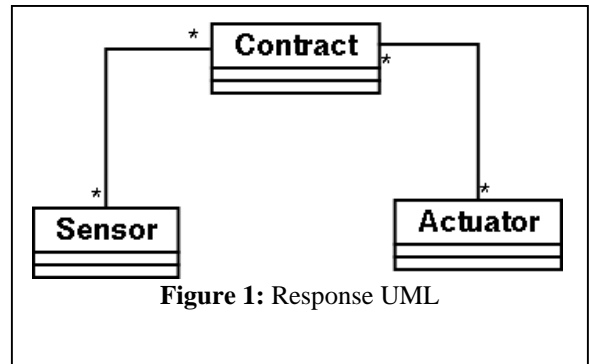


Figure 1: Response UML

- *Actuators* carry out actions/responses at the command of the contracts.

Sensors can report to multiple contracts and a single actuator may be appropriate for different contracts and be shared between them all. These three components may have special consideration placed upon them by their context and are specialized in the patterns presented below.

**Dynamics.** The following collaborations occur in response patterns:

- Sensors gather the data and provide it to the contract or contracts
- The contract takes in sensor information and decides upon a course of action.
- Once a decision has been made, the contract calls upon actuators to carry out the plan.

Again, the patterns presented below will specialize some of these interactions for their specific needs.

**Implementation.** Implementing the patterns consists of the following steps:

1. *Determine what to measure.* Find out the information of interest in the system.
2. *Implement sensors.* Package the state so that it is available to the contracts. Providing state can be done in many ways: push, pull, or a hybrid approach. Each may be useful in some situations. Push ensures that up-to-date information is always available but may consume resources sending unused information. Pull may not deliver data in time but never sends information unnecessarily. A hybrid, Observer[3], approach can be used as a middle ground.
3. *Create contracts to determine responses.* Once the measures of interest have been set, see what combinations or data or states require responses. Encode this information into a contract that can determine what action is required. A contract looks at the data provided by the sensors and

decides what action to take. Contracts may make use of finite-state-machines, control theoretic approaches, or other techniques, when coming up with a response.

4. *Create actuators to carry out responses.* Once the contracts have decided upon a course of action, the actuators are responsible for carrying it out. These responses are encapsulated within actuators and are fired on the contract's command.

Creating actuators that can fire asynchronously allows other sensor/contract/actuator loops to be analyzing and executing while others actuators are carrying out their responses. If actuators cannot be called asynchronously extra sensors can be used to monitor the state of the actuator in question and coordinate between actuators.

A first attempt at a system to react to sensor input may mix the concerns of reading the data, deciding what to do, and carrying out the action. The steps outlined above help to separate concerns and makes the system more extensible and easier to understand. For example, keeping the collection of data separate from the contracts means that contracts do not necessarily need to worry about other contracts and do not have to worry about passing data on to these other contracts.

**Consequences.** The reaction patterns in general offer the following **benefits**:

- *Separate sensing of a problem from acting upon it.* Separating the concerns of noticing a problem and acting upon it allows sensors or actuators to be independent of one another.

**See Also.**

The Contract and Snapshot pattern [2] provides insights into creating contracts as well as collecting and reporting sensor data.

## Rapid Local Response

---

---

The *Rapid Local Response* pattern provides the ability to make quick decisions about a given situation. It separates gathering information from acting upon it in a way in that allows quick responses and extensibility.

---

---



**Examples.** Consider a scheduler for real-time tasks on a platform that is part of a larger DRE system. A fixed number of tasks are considered hard real-time and are statically scheduled. Other tasks might come and go, are dynamic in number and are considered soft real-time. Another example is a disaster response scenario, with multiple autonomous surveillance systems working in conjunction with situation assessment centers and rescue/medical personnel. The number of communication sources and targets will change as vehicles and personnel enter and leave. The number of tasks and operations will change as situations emerge and get resolved. The various constituent nodes will have certain tasks, such as navigation, sensor control, or track processing, that are static in number and can be statically scheduled. Other tasks and operations, such as collaboration with other nodes, data processing, and coordination of rescue operations, might be more dynamic in number and in nature (e.g., amount of data and processing involved). The scheduler cannot perform full analysis and schedule synthesis each time conditions change because of the time and resources involved. However, not adapting can result

in significant numbers of tasks and operations being missed when conditions are overloaded or resources being wasted as conditions become unloaded.

In another example, consider a command and control system that is part of a networked, theater-wide defense system. Such a system, and all of its mission-critical operations, is susceptible to cyber-attack. The first action an attacker might take is to do a port-scan of the target host and then execute a specific attack using the results of their reconnaissance. Even if an intrusion detection system (IDS) is running on the attacked host, an automated follow-up attack may happen before an administrator has time to analyze the logs and see the signs of the impending attack. The attacker may have enough time to cover their tracks and erase the logs before anyone has an opportunity to see what is truly occurring. Autonomic attack responses, used to limit the damage of attacks on critical systems, may be the only viable survival strategy in the face of automated attacks. Such responses will need to be based upon local information (i.e., the symptoms of the attack) and be rapid (to counter the automated attack). Any response that requires significant analysis, non-local information, or human-in-the-loop response is likely to be too slow to counter the effects of an automated attack.

**Context.** An application where decisions on operating mode or responses must be made quickly.

**Problem.** Many DRE applications need to make quick decisions when presented with external inputs. A decision needs to be made before all the consequences of a given reaction can be analyzed. If these reactions are not taken quickly, they may no longer be appropriate or may have missed an opportunity.

Solving this problem requires the resolution of the following additional forces:

- Responses must be carried out quickly or they will not be useful. Wholesale mode changes and analysis are too slow.
- Only local information is available to use in making decisions
- Appropriate responses need to be available for a variety of situations in which the system may find itself.

**Solution.** Create a system that can quickly assess the current situation and take action based on limited knowledge. These instinctive responses can be used to save the system from acute problems that do not leave time for detailed analysis.

In Detail: Define a set of local *sensors* that can be used to quickly measure the system state. Define a set of *actuators* that can respond to events. Tie the two together using *contracts* that quickly reach a decision on how to act.

**Structure.** The Rapid Reaction Pattern has the expected three participants from the template. In this case, the contracts are characterized by simple rules and choices. A contract in this context may be as simple as a collection of if-statements that looks at a few sensor values.

**Dynamics.** The collaborations discussed above occur with the caveat that they must perform their role quickly. From start to finish there is no time to collaborate remotely and time is of the essence. Undue delay in any step may make the reaction useless.

Figure 2 shows an example interaction of rapid local response. The contract desires data from three sensors, but data from one sensor arrives too late and therefore cannot be used to carry out the reaction. The contract carries out its reaction with the data available.

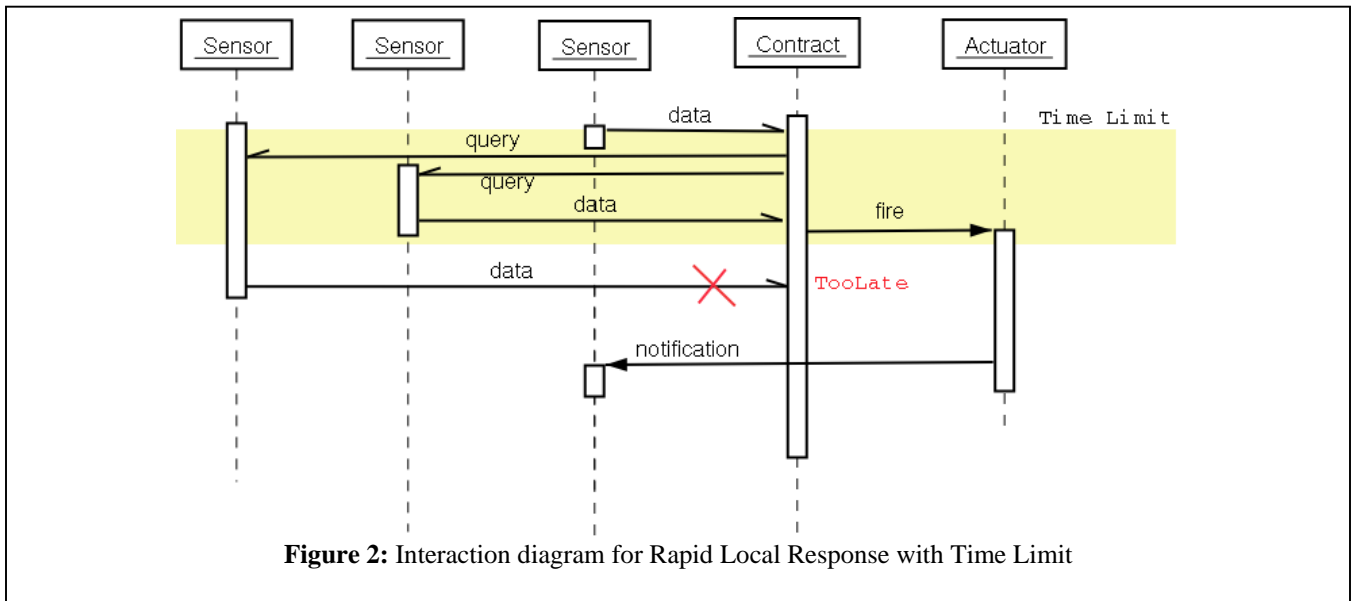


Figure 2: Interaction diagram for Rapid Local Response with Time Limit

**Implementation.** Implementing the Rapid Reaction Pattern consists of the above specified steps with the additional constraint the speed is critical. The sensors need to be ready to provide their state through a method call to a contract or by notifying the contract directly when state changes. Contracts need to ensure that they do not wait on inputs that may not arrive in time. Splitting concerns using multiple contracts can be used to keep any single contract from becoming too complex and can also make them independent of one another.

Rapid reactions work best when implemented as reactive components of a system. As the sensors notice events or state changes they immediately notify the contracts, which then fire the actuators. It is possible for the contracts to poll the sensors. However, if polling is not done frequently enough, by the time the contracts have the opportunity to analyze their new information, it may be too late to effectively fire the actuators.

**Examples Resolved.** There are a number of strategies that a scheduler can use to fairly deal with additional tasks. For example, the rates of other tasks can be adjusted downward (within an acceptable range) to accommodate additional tasks. The scheduling strategy can be changed, e.g., to earliest deadline first (EDF) or maximum urgency first (MUF). A load shedding strategy can be triggered. Which-ever reaction is chosen, the scheduler needs to recognize when the load has increased (i.e., the number of tasks has increased) and rapidly trigger a response, before too many events have been missed. This can be implemented by using the Rapid Local Response pattern to monitor missed deadlines and new tasks, to choose a rapid response, and trigger it.

In the second example, sensors can be installed to monitor the symptoms of attacks, such as port scans (precursor to many attacks) and resource usage (a symptom of many denial of service attacks). These

sensors wake up a contract which triggers actuators, e.g., a firewall which blocks all future traffic from the sender of the port-scan. Later an operator may notice the port-scan in the logs but in the mean time the system has taken steps to mitigate the danger posed by the attacker.

**Known Uses.** The Sidewinder G2 Firewall from Secure Computing makes use of rapid local reactions via its Strikeback [4] capability. The firewall can associate *event responses* with alarm events such as a network probes or excess traffic. These strikeback responses occur as soon as the event is noted.

The WSOA project [5] handles dynamic CPU load in a networked avionics platform by assigning ranges of acceptable rates to soft real-time tasks. As the system recognizes over- or under-utilization of the CPU, by monitoring queue lengths and missed deadlines, it changes the rates of tasks to compensate.

Many Real-Time application need to make use of the Rapid Local Reaction pattern. When memory usage becomes too high, unless something is done to make more memory available, the whole system may become unusable.

**Consequences.** In addition to those specified above, the *Rapid Reaction* pattern offers the following **benefits**:

- *Allows applications to quickly respond to changes.* This keeps applications running in situations where lack of decisive action would be detrimental.

The *Pattern* also incurs the following additional **liabilities**:

- *Locally, but not globally optimal solutions.* Given time to analyze the situation the quickest answers may not be the best ones. A solution that works well for one portion of a distributed system may negatively impact other parts of the system.
- *Unforeseen interactions.* Sometimes the lack of analysis done before making a decision may have adverse effects on the rest of the system, even locally. Thrashing, when many applications attempt to put their data in main memory, is a good example of this.

## Coordinated Response Pattern

---

The *Coordinated Response* pattern provides for a distributed system to make decisions taking into account the desires of multiple system members.



---

**Example.** A group of autonomous vehicles (UAVs) is providing surveillance in a disaster response situation. The environment is highly dynamic, with additional surveillance, command and control, and rescue vehicles and personnel entering the theatre, with victims and sites being detected and responded to, and all of them sharing communication resources that may not be reliable. The UAVs (and other participants) must coordinate to share the available bandwidth and to allocate it to the most important

information (such as detected victims or critical command/control messages). This must be done dynamically to accommodate the changing situation.

**Context.** A distributed application where decisions cannot be made by a single system, but must be made in the context of multiple members of the system.

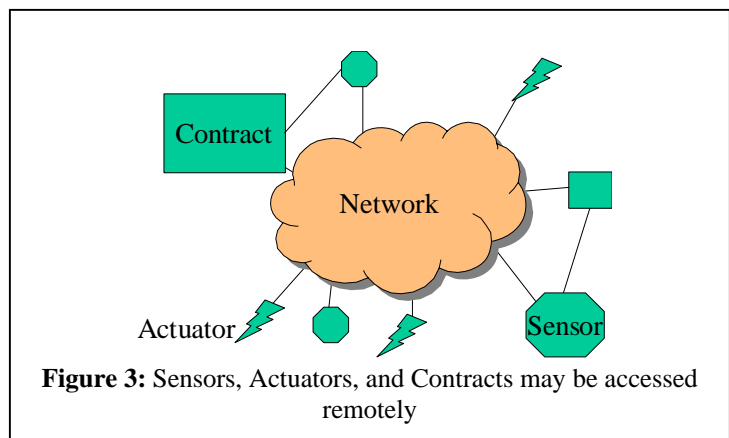
**Problem.** Making decisions in DRE systems often requires agreements among participants when actions are taken that affect the system. Participants need to work together for the good of the system but do not always know what is happening to one another. This conflict is especially troublesome when changes need to be made or events need to be reacted to. If the changes are not coordinated they may be working to opposite ends. Coordinating between participants requires resolving the following additional *forces*:

- No one member knows enough to make the decision without help. Each member may have information that is necessary to make the best decision.
- Participants will pursue competing goals if left to their own devices.
- Information becomes available at different times. All the participants do not learn information at the same time, or may be in a position to learn of it at all. Yet, this information may be necessary to coordinate.
- Time is available to coordinate. There is enough time available for participants to communicate with one another. However, they cannot wait forever.
- Control of individual participants may be limited. While large scale decisions can be decided among participants some decisions will still be made locally.

**Solution.** Before making a decision, share information among participants and come to a conclusion on what is best for the system as a whole. When a decision needs to be made that requires information from multiple places in the system, gather this information and make a decision based on the available information about the system as a whole. Decouple gathering information from decision making, and from carrying out decisions.

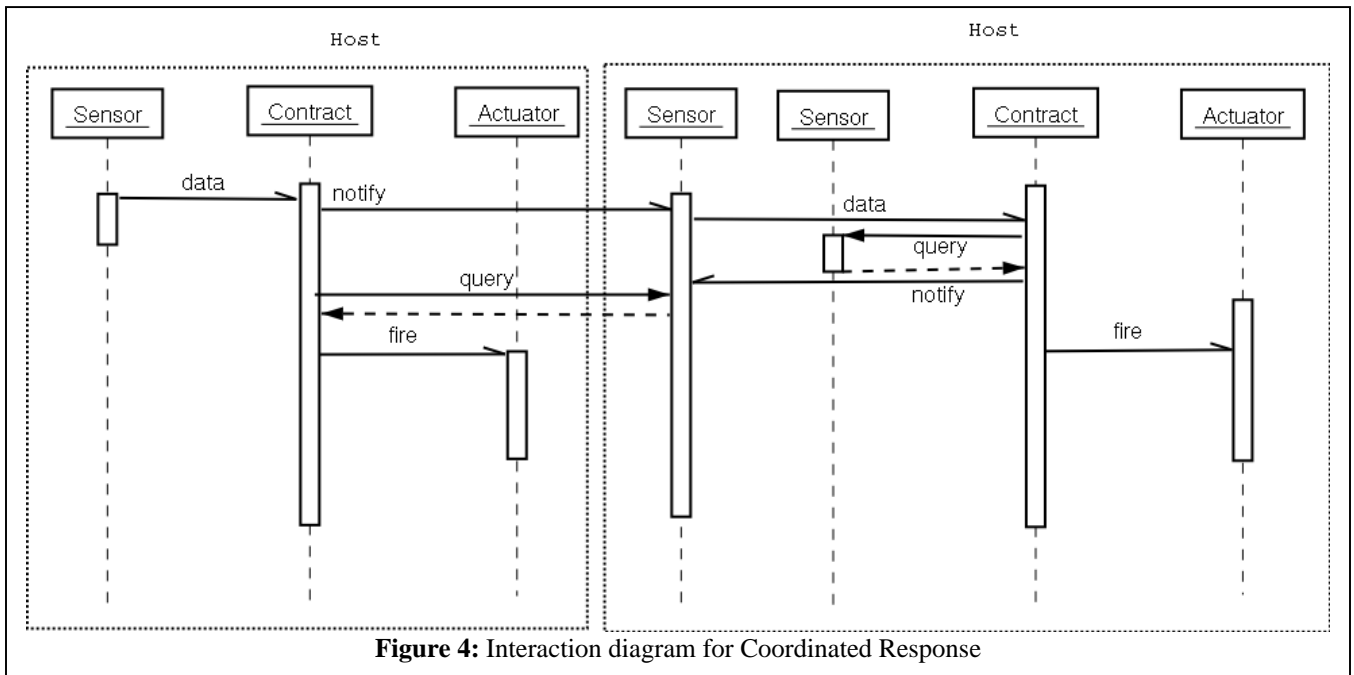
In Detail: Define a set of *sensors* that measure interesting system state and can be queried remotely. Define *contracts*, which take as input data from the sensors and decide the best course of action for the system as a whole. Define *actuators*, which carry out decisions when commanded by the contracts.

**Structure.** The Coordinate Response pattern uses the template structure discussed above. The sensors in this case allow information to be queried remotely so that it is available to interested remote parties, as shown in Figure 3. Special sensors may also be used to hold data about the current collaboration status or to inform contracts of decisions made by other contracts.



**Dynamics.** The following special collaborations, shown in Figure 4, occur in the Coordinate Response pattern:

- The contract takes in the new information and if necessary queries other sensors in order to carry out a coordinated response.
- Once a decision has been decided upon, the contract calls upon an actuator to carry out the plan. This actuator may need to interact with other sensors on remote hosts that may signal other contracts to take action.



The contracts used in Coordinated Responses are more complex than those used in Rapid Local Reactions. They may coordinate with one another through sensors that provide information about contracts and may need to negotiate with one another when there is no obvious best solution. The middle sensor in Figure 4 shows an example of using a sensor to share data between contracts.

**Implementation.** Implementing the Coordinated Response pattern consists of the previously mentioned steps with wrinkles added due to coordination: gathering remote data and resolving competing goals.

Contracts need to gather information from all the relevant sensors. However, some information may be unavailable. The contract needs to take this into account. At some level, if enough information is not available, a contract may choose to not make any changes. However, other times a contract may have to make do with the information available. Contracts often need to set a time after which they will wait no longer and will act on the available information. While the timing constraints are not as severe as with the *Rapid Response* pattern they cannot be disregarded.

Deciding between competing goals is another complex issue. Contracts may need to set rules or priorities used to coordinate and mediate between goals [6,7]. In order to carry out coordinated actions, actuators may need to make calls on remote systems. They will set remote sensors, whose change will trigger contracts, which in turn fire their own actuators.

Decoupling the contracts from the sensors is especially helpful in distributed systems. By providing a facade for the sensors, the contracts do not need to deal directly with the mechanisms needed to measure state. This simplifies contract creation and also keeps redundant code, which may be making requests over the network, from being included in multiple contracts and also provides opportunities for caching of data. Splitting out actuators from contracts allows changes to be made in the responses without needing to change every contract.

**Example Resolved.** The UAVs can make use of the Coordinated Response pattern. A coordinated contract (or set of contracts) will monitor the number of UAVs and predict the amount of available bandwidth. It will calculate the amount of bandwidth available for each individual UAV and push that information to a contract on each UAV. This contract will decide the best way for the UAV to use the bandwidth allocated to it (e.g., lower rate of imagery, smaller or compressed images, lower fidelity images). When a disaster command center operator notices that a UAV is surveying a situation that requires response, he indicates that to the coordination contract, which reallocates bandwidth to assign a higher amount to the UAV (or UAVs) surveying the situation. The bandwidth available to the other UAVs is scaled back to compensate and communicated to them.

#### **Variants.**

The way in which decisions are made can vary in the Coordinate Response pattern. Either one member of the system can make the decision and pass its conclusion on to the rest of the system or multiple participants can come to the same conclusion independently. When a single decision maker is used the action taken will always be consistent. Even if some participants are not able to gather all the necessary information, the decision maker only needs to share its conclusions. If these conclusions are large or complicated they may be difficult to distribute, in which case a distributed decision may be a better choice.

A distributed decision is also possible. In this case each participant gathers the input data itself and decides on the correct response or coordinates with the others to come to a conclusion. Each participant trusts the other participants to decide upon complementary conclusions and carry them out correctly.

**Known Uses.** The Mosix[8] system makes use of Coordinate Responses when processes are migrated. Processes are moved from one node in a computing cluster to another in order to better balance the cluster's load.

The ITUA [9] system uses a Coordinated Response to kill faulty replicas. Each manager reports failures. When a replica's manager sees that a majority of the managers believe that a replica is faulty, the manager will kill the replica in question.

The UAV application in DARPA's PCES program [10] uses coordinated contracts to divide available bandwidth between a dynamic number of UAV imagery sources. In this application, the coordinated contracts use knowledge of the available bandwidth, number of UAVs, and the importance of what a UAV is observing to decide how to allocate the bandwidth among the UAVs. In addition, the contracts on each UAV use information about how the imagery from the UAV is to be used and the amount of bandwidth allocated to decide how to provide the imagery.

Traffic through an intersection is a real world example of Coordinated Response. Drivers watch traffic lights and the other drivers to make decisions if they should go through intersections. However, if an ambulance is coming they pull over and let it pass, even if they have a green light and no one in front of them.

**Consequences.** The *Coordinated Response* pattern offers the following additional **benefits**:

- *Responses will react for the good of the system.* Since the decisions use information from a group of nodes or a subsystem, a locally helpful response that hurts the larger collection can be anticipated and a better alternative can be carried out.

The *Pattern* also incurs the following additional **liabilities**:

- *The responses may be slow.* Communicating between participants takes time and this delay may make responses less effective.
- *Sharing information may be a security problem.* The wider information is disseminated the more opportunities there are for bad uses to be made of it.

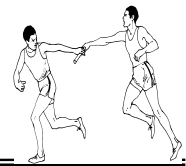
**See Also.**

- The Resource Rationalizer pattern language [6] provides information about scheduling.

## Local and Coordinated Response Compound Pattern

---

The *Local and Coordinated Response* compound pattern is an architectural pattern that allows rapid reactions to be taken and then checked using coordinated responses.



**Example.** The Rapid Reaction pattern introduced the example of blocking the source of a port-scan. Imagine now that an attacker knew that the response to a port-scan was to block all traffic from the "attacking" host. By forging the sender on the port-scan packets, the attacker could get the system to block important traffic from legitimate sources. The rapid response is not good if it causes legitimate traffic to be blocked, but on the other hand taking time to figure out that blocking is unnecessary may leave enough time for a true attacker to complete an attack when blocking should have been done.

**Context.** A distributed system where quick responses are necessary but where other, better, responses may be found after analyzing a more complete picture of the situation.

**Problem.** Reacting to events is difficult. If a response is not quick enough bad things may slip through. Taking time to think may give a better answer in the long run. Both local/rapid and more coordinated responses are needed, often at the same time. The following forces come into play:

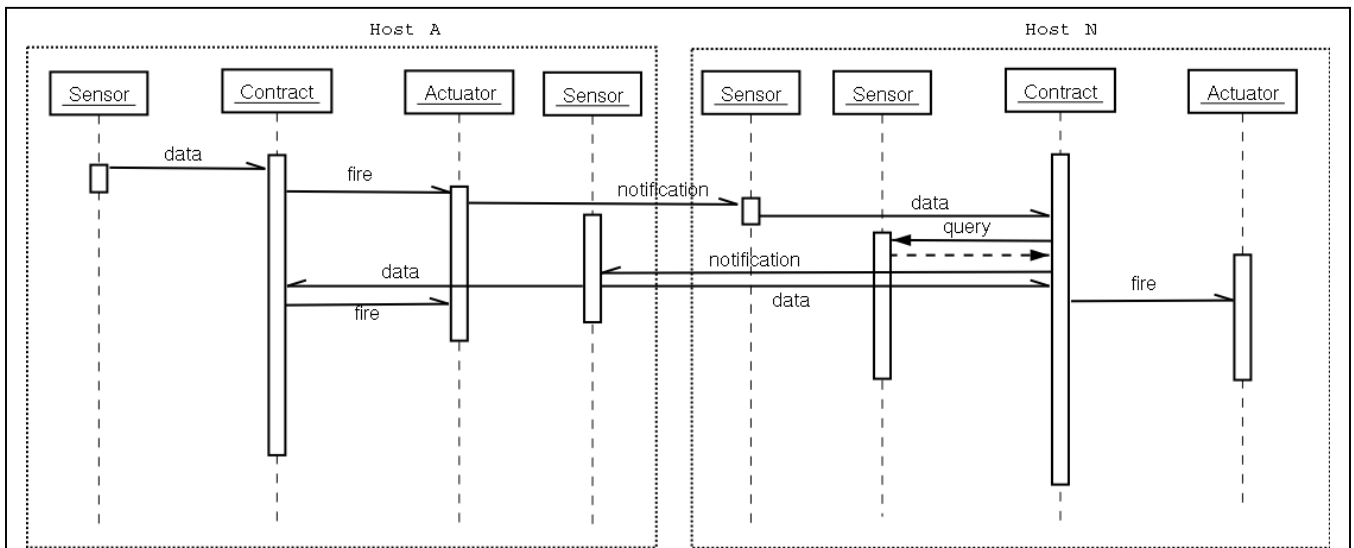
- Some aspects of the system benefit from rapid/local responses
- Some decisions benefit from slower, coordinated responses
- A rapid response needs to be reconsidered in light of coordinated responses.

**Solution.** Combine rapid/local responses with coordinated responses. By using a combination of local responses with slower, coordinated responses, an instinctive reaction can provide a first response that can be superseded at a later point by a more informed response.

**Structure.** This pattern again makes use of sensors, contracts, and actuators as described in the previous patterns.

**Dynamics.** Responding to events using local and coordinated responses consists of the following steps, shown in Figure 5.

- When events first occur execute knee-jerk rapid reactions. This is a best effort attempt to keep the system in a good state.
- Distribute the information that led to the rapid reaction and the reaction taken. This provides input for crafting a more coordinated response.
- Decide what the best course of action is given all the available information. The rapid reaction buys time to consider the best course of action. Use this time to make a coordinated decision.
- Carry out the coordinated response. Once the coordinated response has been decided, carry it out. This may involve rolling back rapid changes that failed to take some information into account or produced undesirable results.



**Figure 5:** Interaction diagram for Local and Coordinated Response

**Implementation.** There are a few main activities to be done in implementing Rapid and Coordinated Responses.

- Decide what rapid responses are appropriate and implement them as described in the Rapid Local Reaction pattern. Create the actuators bearing in mind that a later coordinated response may need to undo the changes the actuator initiated. For example, shutting down a system may be too drastic but an overly strict firewall policy that is expected to be rolled back may be a good choice.
- Implement Coordinated responses as described in the Coordinated Response pattern. The actuators for coordinated responses may need to undo changes put in place by the rapid local responses.
- Hierarchies of responses may be created so that concerns are separated.

**Example Resolved.** In order to protect from a true attack, led off by a port-scan, a rapid response will block traffic from the sender of the port-scan. However, to mitigate the risk of an attacker forging the sender a coordinated response will be started after the rapid reaction. This coordinated response will note that the attacker forged the address of an important server and will roll back the firewall blocking rule put in place by the rapid reaction.

**Known Uses.** The APOD [11] project uses the Local and Coordinated Response pattern. File-system rapid reactions replace modified files and report this to the rest of the system where it may be decided to stop communicating with the effected system.

The Akamai Distributed Content Delivery System [12] is filled with local and coordinated responses. Network conditions are monitored in order to optimize queries via coordinated responses. A request is routed to the best server out of many possibilities based upon a coordinated network view. When a steaming data provider fails a rapid reaction replaces the failed server without noticeably interrupting the data stream.

Spelling correcting software often exhibits this behavior. A commonly misspelled word, such as ‘teh’, may be automatically corrected, while a more complex misspelling may be simply highlighted, waiting for the user to come and fix it.

**Consequences.** The *Local and Coordinated Response* compound pattern offers the following additional **benefits**:

- *Rapid responses can protect a system quickly, allowing time for coordinated responses.* The rapid responses buy time for the coordinated responses to find a more informed course of action.
- *Rapid responses that are found to be faulty are not perpetuated.* Rapid reactions can be undertaken without as much worry about their long-term effects on the system. The coordinated responses can tidy up quick and dirty responses.

The *Pattern* also incurs the following additional **liabilities**:

- *Performance.* As more introspection is added to the system there is less power available to carry out the main task.

- *Rapid reactions may be at odds with coordinated ones.* Implementers need to be careful that the rapid responses are in synch with the coordinated ones so that a coordinated response does not trigger a rapid reaction that then sets off another coordinated response in a loop.
- *Local responses may still be troublesome, but not for as long.* By allowing rapid responses into the system uninformed decisions will still get made.

**See Also.**

- The patterns presented in [13] also deal with resource management, monitoring, and control.

**References**

[1] Boyd, John R. From the unpublished collection of briefing slides entitled "A Discourse on Winning and Losing." Maxwell AFB, AL: Air University Library, Document No. M-U 43947, August 1987.

[2] Joseph Loyall, Paul Rubel, Michael Atighetchi, Richard Schantz, and John Zinky. "Emerging Patterns in Adaptive, Distributed Real-Time, Embedded Middleware". *OOPSLA 2002 Workshop - Patterns in Distributed Real-time and Embedded Systems*, November 5, 2002, Seattle, Washington.

[3] Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] Secure Computing. Strikeback: The Sidewinder G2 Firewall Strategy for Intrusion Detection and Response. Retrieved July 8, 2003 from [http://www.securecomputing.com/pdf/swind\\_strikeback\\_sb.pdf](http://www.securecomputing.com/pdf/swind_strikeback_sb.pdf)

[5] Joseph Loyall, JM Gossett, Christopher Gill, Richard Schantz, John Zinky, Partha Pal, Richard Shapiro, Craig Rodrigues, Michael Atighetchi, and David Karr. "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications". *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21)*, April 16-19, 2001, Phoenix, Arizona.

[6] C. Gill, D. Niehaus, V. Subramonian, L. DiPippo and V. Wolfe "Resource Rationalizer: A Pattern Language for Multi-Scale Scheduling". *Proceedings of the 9th Annual Conference on the Pattern Languages of Programs*, Monticello, Illinois, September, 2002.

[7] Jürgen Lind, "Patterns in Agent-Oriented Software Engineering". Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002.

[8] Amnon Barak and Oren La'adan, "The MOSIX Multicomputer Operating System for High Performance Cluster Computing," *Journal of Future Generation Computer Systems*, 13(4-5), March 1998, 361-372.

[9] M. Cukier, T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal. R. Watro, and J. Gossett. "Providing Intrusion Tolerance With

ITUA". Supplement of the 2002 International Conference on Dependable Systems and Networks, June 23-26, 2002.

[10] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Christopher D. Gill, Balachandran Natarajan, Craig Rodrigues, Joseph Loyall, and Richard E. Schantz. "Total Quality of Service Provisioning in Middleware and Applications". *The Journal of Microprocessors and Microsystems*, Elsevier, vol. 26, number 9-10, January 2003.

[11] Michael Atighetchi, Partha Pal, Chris Jones, Paul Rubel, Richard Schantz, Joseph Loyall, and John Zinky. "Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience". *The 3<sup>rd</sup> International Workshop on Distributed Auto-adaptive and Reconfigurable Systems*, in conjunction with the 23rd International Conference on Distributed Computing Systems, Providence, Rhode Island, USA. May 19-22, 2003.

[12] John Dilly, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, Bill Weihl. "Globally Distributed Content Delivery". *IEEE Internet Computing*, Vol 6, number 5, September/October 2002.

[13] Toni Marinucci, Lonnie R. Welch, Michael W. Masters, and Paul V. Werme. "A Pattern Language for Engineering Dynamic Real-Time Applications". *Proceedings of the 9th Annual Conference on the Pattern Languages of Programs*, Monticello, Illinois, September, 2002.