Event-Based Architecture

What is event-based architecture?
Event-based architecture is an architectural design that uses the production and consumption of events to control behaviour. When a specific component has finished running, instead of directly calling a function of another component to incite additional actions, it would instead produce and propagate an event out via an event bus to other components, which will then react by performing some actions.

Does event-based architecture have its own vocabulary for its components and connectors?
Components:

- Event generators[1]
    - Components that generate events
- Event consumers[1]
    - Components that consume events
    - May react to the event with some functionality and/or transform/forward the event along a channel

Connectors:

- Event Bus(es)[1]
    - Medium on which events are propagated
    - Consumer components consume events off of the event bus
    - Exclusively knows correct distribution of events

Does event-based architecture impose specific topological constraints?
Components must communicate with each other via events on the event bus. Components do not communicate with each other directly.

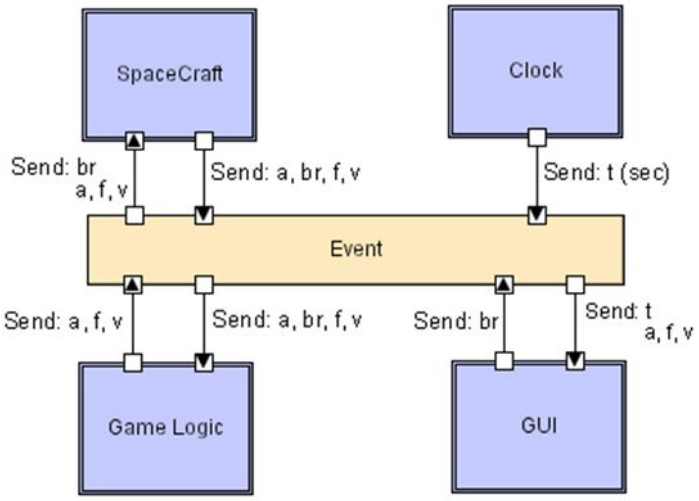

Image taken from the "05_Architectural_Styles" slides that are available at http://www.softwarearchitecturebook.com/resources/.

Is event-based architecture most applicable to specific kinds of problems?

- Event-driven applications
    - User interfaces
    - Hardware events
- Distributed applications[2]

Does event-based architecture engender specific kinds of change resilience?

- Changing event names or types requires changing consumers and generators
    - Changes to event name require changing every consumer/generator for that event
    - Changing type of event requires changes to how consumers handle the event and how producers generate the event
- Tightly coupled to the semantics of the underlying event schema and values

Does event-based architecture have any specific negative behaviours?

- Message passing may not be as efficient
    - Extra layer of abstraction
    - The event bus is a shared limited resource
- Event bus may become a bottleneck
    - Many concurrent generators
    - Many consumers for a single event
- Harder to follow the control flow
    - Many asynchronous calls occur simultaneously and cascade out
    - Possible for consumers to generate new events, extending the chain

Does event-based architecture support/inhibit specific NFPs?

- Highly scalable[1]
    - Consumers that receive the event are able to react concurrently, no need to wait for each one to finish before another consumer can run.
    - Communication only done via event bus (channel), can add additional functionality by simply adding another event consumer which will react to a specific event sent through the bus.
    - Event consumers be on a difference process or machine
- Loose coupling between components[2]
    - A component that emits events will not know the implementation of, or even the existence of any event consumers. Likewise, event consumers are only concerned about reacting to an event as soon as it occurs.
- Easy to evolve[1]
    - Enabled by loose coupling (isolated components interact using a common interface); components can evolve independently without changing interface

Sources

[1] *Event-Based Architecture*. “05_Architectural_Styles.pdf”. Retrieved from: http://www.softwarearchitecturebook.com/resources/.

[2] *Event-Driven Architecture*. Retrieved from: https://en.wikipedia.org/wiki/Event-driven_architecture