

# Modeling and Causal Event Simulation of Electronic Business Processes

David Luckham, Antoine Manens, Sumit Bhansali, Woosang Park, Susheel Daswani

**Abstract—** We outline and illustrate an event-based approach to the design and simulation of electronic business (eBusiness) processes. Our goal is to establish that eBusiness processes will achieve business goals prior to production rollout. eBusiness processes are increasingly complex, parallel, and asynchronous systems involving multi-enterprise collaboration. Therefore new simulation and testing techniques aimed at cutting the costs of process errors are increasingly important.

Our approach simulates behavioral models of processes expressed in an event-based modeling language called Rapide. These process models embody the business rules and policies of the process owners, and are designed according to object-oriented design principles expressed in Rapide. The Rapide simulator utilizes a unique approach to event-driven simulation of processes by tracking the causal relationships between events during simulation. This allows the root causes of errors to be discovered easily during simulation. Design errors whereby a process design will not meet business or performance goals, can be detected early in the lifecycle and corrected before the processes are put into production. Design constraints (i.e., formulations in Rapide of business policies and rules) can be used to automatically detect violations of business rules at simulation. Also they can be incorporated into the production versions of the processes and used to detect errors in real-time in actual operations.

This paper illustrates our approach using an example of modeling and simulating a new process designed by partnering enterprises to execute over an ISO 15022 financial trading network. The new process enables collaboration between their existing business processes. This approach can be applied using a causal event simulator for any object-oriented process modeling language.

## I. INTRODUCTION

Business processes have outgrown traditional linear workflow.<sup>1</sup> As a result, more sophisticated process languages are becoming industry standards (e.g., BPML [1], WSFL [3] and WSDL [4]). A primary cause is the growing trend to

<sup>1</sup> A Web search, “workflow standards” or “Workflow coalition” will yield abundant references.

electronic business-to-business (B2B) collaboration using the Internet and eMarketplaces (see, e.g., the eSpeak platform [2]). Electronic business processes are no longer sequential flows of activities related to document processing. The new generation processes incorporate complex, parallel, asynchronous decision-making. These processes work at web speed with far less human involvement than before.

New design and development methodologies are needed to deal with the new generation of complex eBusiness processes. Present day process lifecycle development technology separates design, test, and deployment phases. This means that a process’s behavior is seldom well understood until it has been fielded. Process modifications are largely devoted to fixing the process to deal with situations that were unforeseen at design (so called *exceptional situations*), but arise during operations. This kind of approach (“fix it when it breaks”) is costly and far too slow in the eBusiness age. Many exceptional situations could be avoided by using more sophisticated design and test methods earlier in the lifecycle. The most costly errors in any system are those errors made at design time that persist throughout the lifecycle.

As electronic commerce develops, business processes will be in a state of *continuous evolution* to meet changes in the business goals of the enterprise, and changes in the eBusiness environment. This will increase the demand for faster, more accurate process development methods.

*Continuous evolution* of processes requires technology to:

1. Improve early stage design methodology, allowing for the complexity of parallel asynchronous processes,
2. Establish at the design stage that process designs will meet their owner’s goals and expectations,
3. Enable continuous monitoring at the operations stage that processes behave in conformance with critical business constraints,
4. Enable correct modification of processes by component-based, plug-and-play methods of component replacement.

This paper outlines and illustrates an event-based approach to designing systems of parallel, asynchronous processes using component-based architectures and causal event simulation. Our approach applies equally well to systems of workflows, or more complex processes. We give an example of a new process that is being designed to integrate existing legacy processes

within each enterprise to effect a business collaboration. The collaboration is to take place across an ISO-15022[11] standard financial transaction network. Such networks enforce type correctness of messages and provide support applications for electronic trade confirmation and settlement.

The cost and time to implement and deploy eBusiness process systems such as our example can be reduced significantly by a relatively small upfront investment to apply powerful event-based design and simulation tools early in the process development lifecycle. Not only are later stage errors avoided, but consensus and agreement is facilitated by early stage simulation and animation of process behavior.

II.APPROACH

Our approach involves using the business rules and policies of the process owner to define behavioral models of the process. The rules and policies are structured into groups that naturally define the properties of individual process components of a process architecture. Architectures are expressed in an event-based modeling language, *Rapide*, [6], [8] which utilizes a unique approach to simulation of models by tracking the causal relationships between events. Rapide also allows models to include constraints expressing requirements. Violations of constraints are detected automatically during simulation. This combination of causal tracking of events and constraint checking enables the root causes of many errors to be found quickly during simulation.

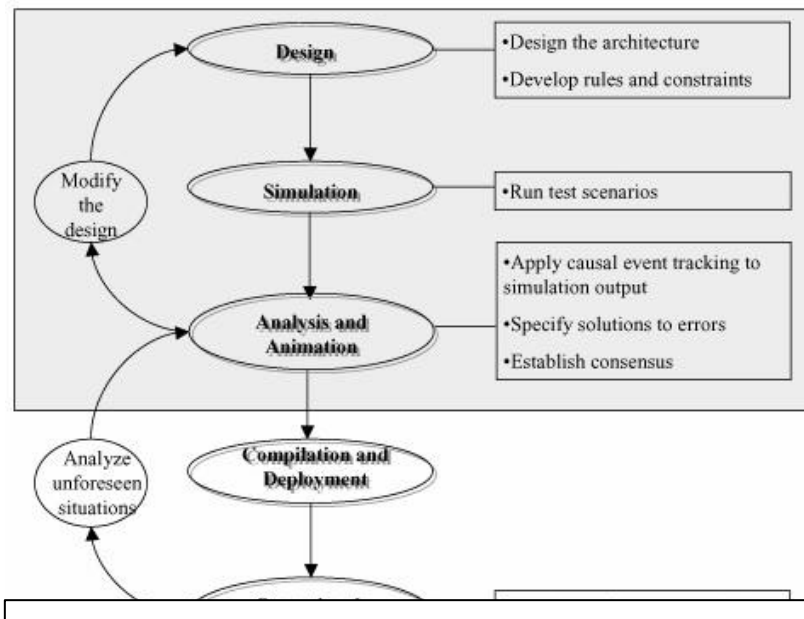
Steps in our approach, as illustrated in Figure 1, are:

1. **Architectural design:** Process designs are expressed using graphical and textual tools as *layered interface communication architectures* in *Rapide* [5], [6], [9].
2. **Structuring sets of business rules:** Business rules are expressed as reactive rules in *Rapide* that define the behavior of individual components in the architecture. Rule sets are structured by associating rules with the

components and connections they apply to.

3. **Specifying business constraints:** Business policies are expressed as behavioral constraints in *Rapide*. Constraint violations are detected during architecture simulations.
4. **Process behavior simulation:** Architectures are simulated using the *Rapide* [6] event-driven simulator that produces causal event simulation output.
5. **Behavior analysis:** Causal event analysis is applied to simulation histories using the *Rapide* analysis toolset [8]. Events of interest in simulation histories such as constraint violations, or unexpected events, can be traced back through causal links to the earliest events in their causal histories.
6. **Process refinement:** The components in the architectural design whose behaviors lead to these events are indicated, and can be modified as needed.

This approach is typical of design and simulation methods used e.g., in hardware design today, except for the introduction of causality relations between events. Similar rigorous methods will need to be applied to autonomous processes in the future.



The design stage, (steps 1 - 6) provides analytic capabilities early in the lifecycle for assessing the correctness and efficiency of the process design, prior to large financial outlays for implementation and deployment. Many early design flaws are caught before they are propagated into deployed systems. Also, the combination of simulation and animation at step 4 is a powerful tool for forming consensus and agreement on process designs between collaborating enterprises. Behavior analysis of simulations (step 6) using causal event tracking is critical in uncovering design errors and effecting rapid revisions in complex process designs. Finally, the formal design rules and constraints used in the process model can be deployed later, by feeding events from actual operation to the same simulation analysis tools, to automatically monitor the deployed process for conformance to the original rules and policies.

In this paper we omit discussion of requirements capture, the task of unraveling the informally described rules and policies that describe the processes, and then formalizing them as reactive rules and constraints.

Our illustrative example is a process negotiated between a merchant bank and a brokerage house to execute financial transactions. Both institutions have internal legacy processes that are not exposed in the collaboration. The collaboration commences when a client places a market order to buy stock XYZ with the bank. The bank forwards the order to a brokerage firm – the broker. The broker is obligated to give each order its best effort execution. The broker may have several strategies to fulfill the order: send the order to the floor of the stock exchange, direct the order to the market maker in charge of the stock, or fulfill the order from the inventory of stocks the brokerage firm owns. These activities are internal to the broker. Once the order is fulfilled, the collaboration continues with an agreed protocol of notification and confirmation messages exchanged between the broker and the bank using an ISO 15022 network. Essentially this is a message protocol. At the successful completion of the protocol, a trade is finalized and cash and stock security are both transferred to a settlement house. Payment is settled for the trade within a certain period of time.

In the model we shall focus on the collaborative trading process and keep settlement very brief. Settlement by itself can be modeled separately and treated as a sub-process of the model.

### III. ILLUSTRATIVE EXAMPLE: COLLABORATING FINANCIAL PROCESSES

*Rapide* is an event-based simulation language in which complex patterns of events are used to express reactive rules governing the behavior of sub-processes and connectors in an architecture. *Rapide* is supported by a compiler, simulator and analysis toolset. This allows us to simulate a complex process by specifying its sub-processes and the connectors by which

events flow between them. *Rapide* also provides a capability, unique among computer languages, to track the causal relationships between events. The analysis toolset uses causality between events in new ways to analyze concurrent and asynchronous systems.

Process architectures in *Rapide* are built out of two kinds of components:

- **Process interfaces:** A process interface represents an activity. It specifies input and output events and a process state. An interface specifies how input and output events are related, and changes in the state of the process.
- **Connectors:** A connector specifies the flow of events between process interfaces as a complex pattern of events.

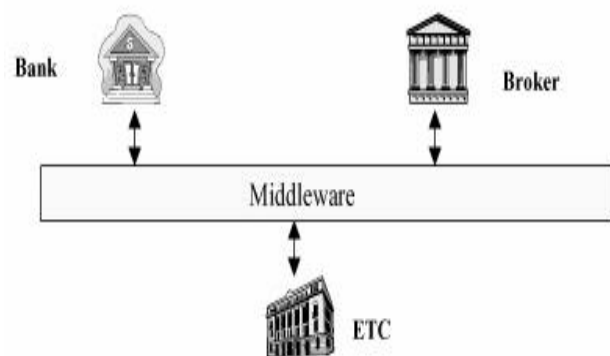
#### STEP 1: INTERFACE COMMUNICATION ARCHITECTURE DESIGN

At *step 1* we design a communication architecture consisting of the types of process interfaces and connectors, and the communication structure. In our simple example, a top-level interface communication architecture consists of interfaces for a merchant bank, a brokerage house, an electronic transaction confirmation facility (ETC) and a main connector for the ISO 15022 financial transaction middleware. (see Figure 2).

Events in this process architecture signify the activities of sending and receiving messages provided by the ISO 15022 [11]. The subset of ISO 15022 message types used in our model, and an overview of how the enterprises agree to collaborate using them, is given in table 1.

All events, except an MT502 initiating or canceling an order, are flowing between the Bank and Broker via the ETC which acts as an intermediary.

Each process interface in figure 2 represents a class of processes. A process class expresses the behavior of a process by reactive rules that define how the process reacts to input events by changing state and creating output events.



**Figure 2: Top level process architecture**

**Table 1: Summary of the collaboration agreement**

MT502: Order to Buy/Sell	The bank (the instructing party) sends this message to the broker (an executing party). It instructs the executing party to buy/sell a given quantity of specified financial instruments. It can also be used to cancel an order.
MT513: Client Advice Of Execution	The broker sends this message to the bank that has submitted a buy/sell order previously. It is used to provide the bank with brief and early information about the deal executed at its instruction. This advice applies to a deal that cannot yet be fully confirmed, for example, because it is a block trade that is to be allocated.
MT514: Trade Allocation Instruction	Messages of this type are sent from the bank to the broker in response to a MT513 message to instruct the allocation of a block trade. Each message contains only one allocation. Several allocations may result from one MT513.
MT515: Client Confirmation of Purchase or Sale	The broker sends this message to the bank in response to a MT514. It is used to confirm the details of a purchase or sale executed by the broker on behalf of the bank. It is also used to provide payment side of the transaction. For each allocation instruction that is sent from the bank, a matching MT515 confirmation is sent back from the broker.
MT517: Trade Confirmation Affirmation	This message is sent by bank to the broker. It is used to positively affirm the details of all previously received MT515 confirmations dealing with an MT502 order.

A process class specifies

- **in events**, the events it can receive and act upon,
- **out events**, the events it can create,
- **local state**, variables local to the interface,
- **behavior**, the reactive rules triggered by patterns of **in** events that specify how it reacts to create **out** events and change local state variables,
- **constraints**, these specify the patterns of **in** and **out** events that its behaviors must conform to.

The behavior and constraints in process classes are specified by reactive event pattern rules written in *Rapide*. Step 2 of our approach involves determining the business rules of the collaboration and structuring them into rule sets by associating them with the process classes.

We now describe some of the rules of collaboration and the process interfaces that must comply with them. Typically, these rules would be agreed upon between the partners, and expressed in some form of document of agreement.

**STEP 2: ASSIGN BUSINESS RULES TO INTERFACES**

1) *Bank*

In order to fulfill its client's requests the Bank uses the services offered by Brokers. Generally, it needs to find the 'best deal' for its client and then confirm that the trade completes according to the client's wishes. Therefore, it

maintains a set of *business rules* dedicated to order fulfillment. Some of these rules are the result of the collaboration agreement.

Table 2 shows one rule, "*respond to advice of execution*".

A reactive rule has two parts: *<event pattern> => actions*; Rules have causal semantics. If the *event pattern* is matched by events input to the rule, then the rule reacts by performing

**Table 2: Rule "Response to advice of execution"**

Elements	Declarations
Variables	String ID; Order OrderInstance;
Event Types	Order is a complex data type with several components. Message MT513 (String ID, Order OrderInstance); Message MT514 (String ID, Order OrderInstance);
Pattern Trigger	MT513 (ID, OrderInstance):
Rule	MT513 (ID, OrderInstance) => [+ rel ~] MT514 (ID, ...);  When a message is received that matches the MT513 Type, this Rule is triggered. One or more (+) events of type MT514 are then created each referring to the ID in the MT513. The "~" means that these MT514 events may or may not be related to one another.
Context Test	Stocks allocated to an account in each MT514 must be part of the OrderInstance trade mentioned in the triggering MT513.
Action	Expression "MT514 (ID, ...)" defines an event MT514 (Trade allocation instruction), responding to the MT513 advice of execution for order ID, to be sent to the partner (in this case, the Broker).
Explanation	The Bank must reply to a MT513 (Advice of execution) with at least one MT514 (Trade allocation instruction) identifying the order ID.

*actions* that can create new events. These new events are *caused* by the events that matched the pattern trigger.

Event patterns

An event pattern expresses combinations of events and relationships between events. Relation operators between events A and B include, A **and** B, A **or** B, A  $\rightarrow$  B (A causes B), A **||** B (A parallel B), A < B (A happens earlier than B) and A ~ B

(A and B can be related in any way). Patterns may include Boolean tests on parameters of events and also variables in the process interface (see [5]or [8] for details of event patterns).

The rule in Table 2 is triggered by an MT513(ID, ...). It creates an instance of the pattern, [+ rel ~] MT514 (ID, ...). That is, a set of one or more MT514 events referring to the trade ID, and related to one another in any way. All of these events are causally related to the MT513 event,

MT513 (ID, ...) → MT514 (ID, ...).

Actions

Whenever the event pattern of a rule matches events in the input to the process interface, the rule reacts by executing two steps. First it substitutes the data values that matched the variables in its pattern wherever those variables occur in its actions. Second, the rule executes the actions. Actions can include changing the values of state variables in the interface and creating new events. Any created events are caused by the events that matched the pattern.

Other examples of the Bank’s collaboration rules are:

- A trade should be initiated with a MT502 (Order to buy/sell) message sent to a Broker.
- If a trade needs to be canceled, an MT502-CANCEL (Instruction to cancel an order) should be sent to the appropriate Broker.
- The Bank should reply to a MT515 (Client confirmation of purchase or sale) with a MT517 (Trade confirmation affirmation).

2) Broker

The Broker needs to execute as many orders as possible to

**Table 3: Rule “Response to Order to execute”**

Elements	Declarations
<b>Variables</b>	String ID; Order OrderInstance; OrderDate Date; Agent FullFiller;
<b>Event Types</b>	MT502 (String ID, Order OrderInstance); MT513 (String ID, Order OrderInstance); CanFullFill (String ID, Agent FullFiller, OrderDate Date, Order OrderInstance);
<b>Pattern Trigger</b>	MT502(ID, OrderInstance, Date, ...) → CanFullFill(ID, FullFiller, Date1, OrderInstance, ... ) <b>where</b> Date1<Date  An MT502 Order is received that causes (→) a Broker’s agent to send a CanFullFill event, and the context test on the date is satisfied.
<b>Rule</b>	MT502(ID, OrderInstance, Date, ...) → CanFullFill(ID, FullFiller, Date1, OrderInstance, ) <b>where</b> Date1<Date ⇒ MT513 (ID, OrderInstance, ...);

<b>Context Test</b>	Date1 < Date: The Date1 specified by the Agent should be earlier than the Date specified in the MT502 order.
<b>Action</b>	Expression “MT513(ID, OrderInstance)” defines an event, MT513 (Advice of execution) that should be sent to the Bank.
<b>Explanation</b>	If the Broker is able to fulfill the order to buy/sell (MT502) by a given Date, it should send a MT513 (Advice of execution).

generate profits. As an example, upon receiving an order to execute a trade (MT502), the Broker should check with its agents that the order can be fulfilled, and confirm to the bank.

The trigger of the rule in Table 3 is a pattern of two causally related events, MT502 → CanFullFill, governed by a context test. When the trigger is matched by events incoming to the Broker, it executes the action.

Another example of a collaborative business rule for the broker is

- the Broker must reply to a MT514 (Trade allocation instruction) with a MT515 (Client confirmation of purchase or sale).

3) ETC

ETC (the electronic trade confirmation house) is a third party necessary for validation of transactions. Generally, the ETC sits between a transaction's participants, intercepts their messages to one another, and validates these messages. The ETC provides validation services to collaborating partners. Examples of business rules for the ETC are:

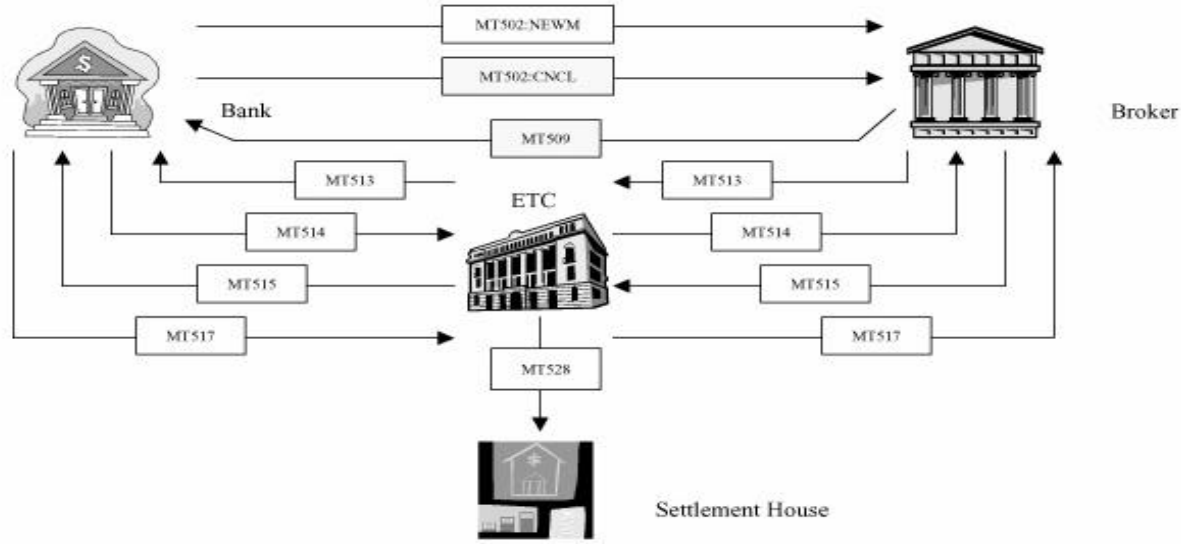
- The messages must have valid formats. The ETC checks that a number of key-fields are present.
- There is a specific trade allocation constraint that must be verified. When an investment bank sends trade allocations to the broker, the total number of shares allocated must be equal to the number of shares purchased by the broker.

4) Settlement House

Generally, the Settlement House brokers the actual movement of capital and stock between transactional partners. Its role would be represented by a sub-architecture. The Settlement House is omitted to keep our model tractable.

**Connectors** in a process architecture are also modeled by interfaces containing event pattern rules. Connector rules specify how events output by some interfaces are input to other interfaces, thus connecting those interfaces. A simple connector rule specifying one type of connector between a Bank and a Broker is:

**Table 4: Simple Connector Event Transport Rule**



Elements	Declarations
<b>Variables</b>	OrderId ID; OrderType T; StockSymbol Sym; Quantity Q;
<b>Event Types</b>	MT502(String ID, OrderType T, StockSymbol Sym, Quantity Q);
<b>Connector</b>	Bank.MT502 (ID, T, Sym, Q) => Broker.MT502 (ID, T, Sym, Q)
<b>Explanation</b>	An MT502 event output by the Bank <b>causes</b> an MT502 event with identical data to be input to the broker.

<b>Constraint</b>	<b>never</b> (MT502 (OrderId) at T ~ MT517 (OrderId) at T' <b>where</b> T' > (T + 3 days) )
<b>Explanation</b>	A pattern of two events, an MT502 initiating an order and an MT517 confirming the same order, should never happen more than 3 days apart. Thus, all transactions should be completed within 3 days.

Connector rules also impose a causal semantics between the events they transport between process interfaces.

### STEP 3: BUSINESS CONSTRAINTS

In addition to the rules and constraints specific to each institution’s interface, there can be *global* constraints that express policies in the agreement that apply to the entire collaborative business process.

All processes in the collaboration must abide by the global constraints. An example of a global constraint in our collaboration model is a timing constraint that applies to any bulk trade.

The timing constraint specifies that any transaction must be executed in three business days. This global constraint, like the process-class specific local constraints, can be specified as a rule in Rapide that governs the flow of events.

**Table 5: Global Constraint on Timing of Transactions**

Elements	Declarations
<b>Variables</b>	String OrderId ; OrderData Data; OrderType Buy, Sell, CNCL ResultType Complete, InProgress;
<b>Event Types</b>	Message MT502(String OrderId, OrderData Data, OrderType T); Message MT517(String OrderId, Result Complete);

As a consequence of Steps 2 and 3, the behaviors of processes and connectors depicted in the architecture of Figure 2 are specified by executable reactive rules. This architecture can be simulated by executing these rules on various event scenarios. Also, the middleware interface has been refined to a set of connectors that specify a detailed transaction protocol whereby events flow between each of the collaborators on the ISO 15022 message bus. This is shown in figure 3. Design constraints formalizing collaboration policy can be used to monitor the event flows and detect policy violations.

### STEP 4: MODEL SIMULATION

The next step is the simulation of the process architecture specified in *Rapide*. The *Rapide* toolset includes a compiler, constraint checker, an animation tool (Raptor), and a simulation analyzer (POV).<sup>2</sup>

The process architecture is compiled into executable code that can be run on different scenarios of events.

1. Constraint violations are detected during simulation and reported.
2. Raptor, an animation tool, displays a real-time animation of message interactions between processes on the architecture diagram of the model (figure 3). Animation is a powerful tool in understanding behavior of the model.
3. The simulation output is a causal event history, or *poset* (a

<sup>2</sup> This toolset is documented on the *Rapide* website,

**Figure 3: Message flow in the collaboration protocol**

partially ordered set of events).<sup>3</sup> The POV poset browser has several features for displaying posets in DAG (Directed Acyclic Graph) format that explicitly show causality between events, for tracking causal histories of events, and for searching for patterns of events in posets.

Poset simulation histories, as we will see, are critical at the behavior analysis *step 5* (Section II). Posets show the causal dependencies between events created by parallel or asynchronous behavior. As the process architecture is simulated on different scenarios, the causal histories are automatically produced. These poset histories can be used to check that the architecture is functioning properly. They are key to understanding how violations of policy constraints, or exceptional conditions in a process execution, are caused. Posets allow us to quickly track the causal history of events that signify such errors. Appropriate modifications can be made to the architecture to allow it to better handle different scenarios. Thus, close examination of the causal histories can be used to refine the process architecture and ensure its correctness.

Many test scenarios were generated in order to cover as many possible situations as possible. Examples of situations tested include:

**Inconsistencies in the protocol event sequences.** These scenarios cover situations outside the scope of ISO-15022 message correctness, where successive steps in the

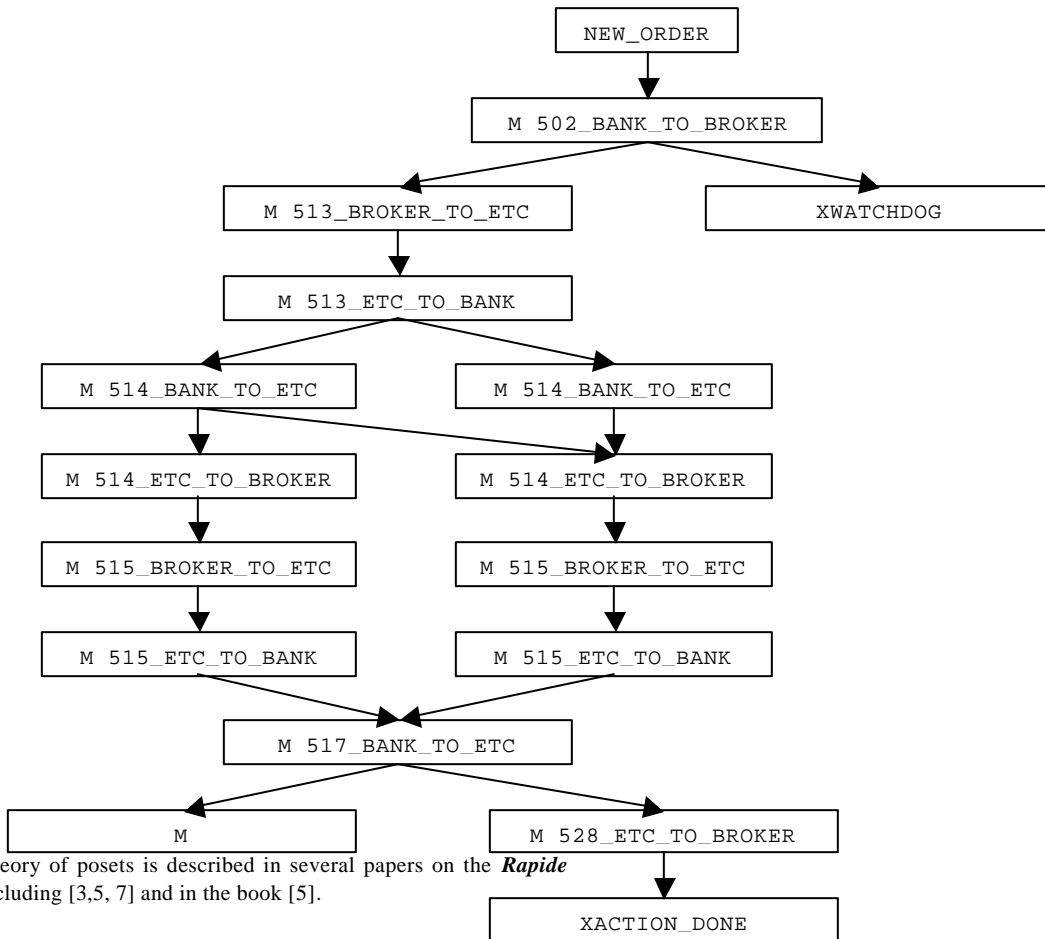
collaboration protocol may not contain data consistent with earlier steps.

- **Race conditions inherent in the collaboration.**
- **Scenarios in the protocol steps leading to violation of the global design constraints.**

**STEP 5: SIMULATION ANALYSIS**

Here we illustrate the use of posets in the analysis of simulation output. Each of the following posets shows a small example of the event execution output by a simulation of the process architecture signifying steps in transactions between the bank and the brokerage.

Figures below show posets in DAG format. The nodes are events, and the arrows signify causality between events. For example in figure 4, the event, MT502\_Bank\_to\_Broker is shown as causing two events, an MT513\_Broker\_to\_ETC, and an XWATCHDOG event. Notice that the two caused events are independent, that is, neither of them causes the other, so there is no causal arrow between them. The first of these events is a message from the Broker confirming execution of the trade requested in the MT502 event. The second event is one that initiates the checking of the global constraint for this particular transaction. In these figures data parameters and timestamps are omitted from the event labels.



<sup>3</sup> The theory of posets is described in several papers on the *Rapide* website, including [3,5, 7] and in the book [5].

**Figure 4: Causal history of a successful transaction**

**1. A Successful Transaction**

Figure 4 shows the poset of a successful transaction. A new order event from a customer to the bank causes an MT502 event from bank to broker. This event causes a watchdog event, initializing a check for the global timing constraint, and also causes an MT513 event from broker to ETC, and from ETC to bank. In turn, the MT513 event causes two MT514 events from the bank to ETC and on to the broker. These MT514 events, allocating portions of the trade to different accounts in the bank, actually happened at different times.

If we look at the causal structure of figure 4, we can see that there are essentially two independent sequences of events caused by the MT513\_Broker\_to\_ETC that are eventually coordinated by the Bank sending a MT517 event. This graphical format for posets gives us a good overview not only of the causal relationships between events, but also the number of independent sub-transactions that are executed.

For each MT514 event it receives from the Bank, the ETC sends another MT514 event onto the broker. The ETC tracks MT514 events to check that a trade is completely and exactly allocated. This checking within ETC invokes accounting applications, which treat events in their time order of arrival. The data placed in an MT514 event by the accounting application depends upon the earlier MT514. Thus the two MT514 events sent to broker not only depend upon the events from the bank, but also have a causal dependency which reflects their order of arrival at the ETC. The second one depends upon the first. A causal rule to add these dependencies between MT514 events depends upon knowledge of the ETC’s internal processes.

The poset shows causal dependencies of subsequent MT515 events and the MT517 event from the bank that finalizes the transaction. The MT517 is caused by both of the

MT515 events, and causes settlement events between ETC and Settlement facilities to happen.

Notice that the sub-transactions on the left side of figure 4 lead to MT528 events from the ETC, which are settlement events to finalize the trade transaction between the Bank and Broker. One of these events triggers a successful XACTION\_DONE event, which turns off the global constraint check that was initialized by XWATCHDOG. There are no violations of constraints, and the collaboration leads to a successful conclusion of a trade requested by the Bank.

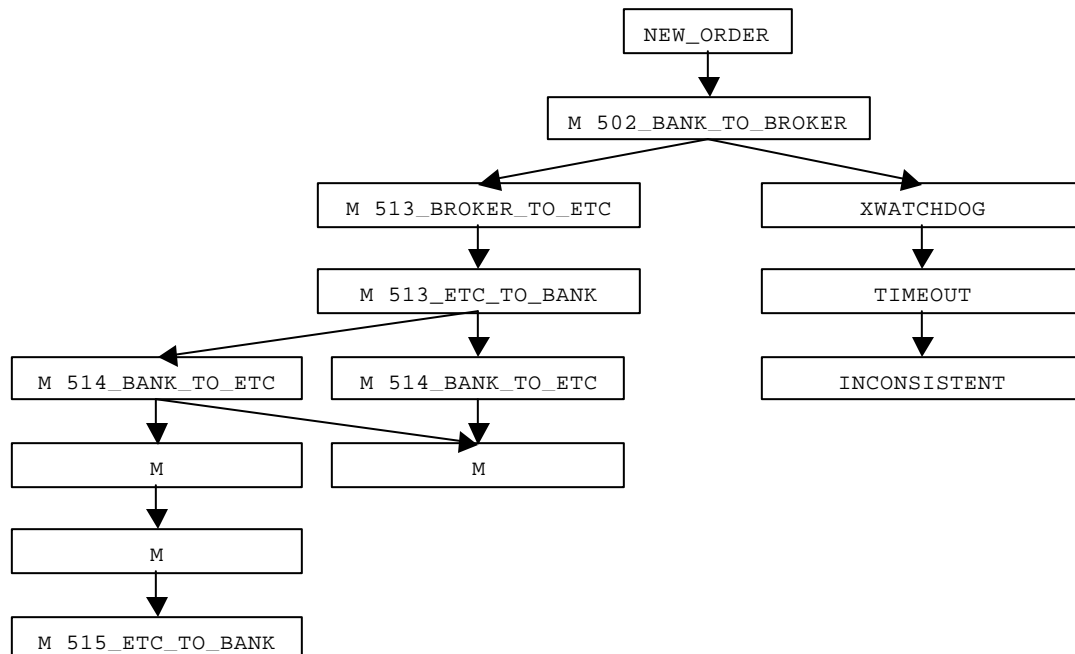
**2. Transaction failed due to Time Out**

Figure 5 shows a poset recording the events in a transaction that did not complete. The activity appears to have proceeded similarly to the first transaction except that:

1. the watchdog initialization event caused a TimeOut event which in turn caused an Inconsistent event report.
2. the broker appears to have failed to respond to an MT514 event.

What caused the TimeOut event? A comparison of the Figure 4 poset with the Figure 5 poset can quickly throw some light on this question. The cause of the time out in Figure 5 is the MT502 event that initialized the XWATCHDOG constraint check. The global timing constraint was violated because the MT502 event did not eventually lead to an MT528 transaction settlement event within the specified 72 hours. The MT528 event is missing.

The deeper question is why that event is missing! We can see from the comparison of Figures 4 and 5 that one of the sub-transactions starting with a MT14\_Bank\_to\_ETC has not been responded to by the Broker. So, the Figure 5 poset indicates precisely which MT514 event in the transaction failed to be processed. A reasonable guess is that there is a problem at the

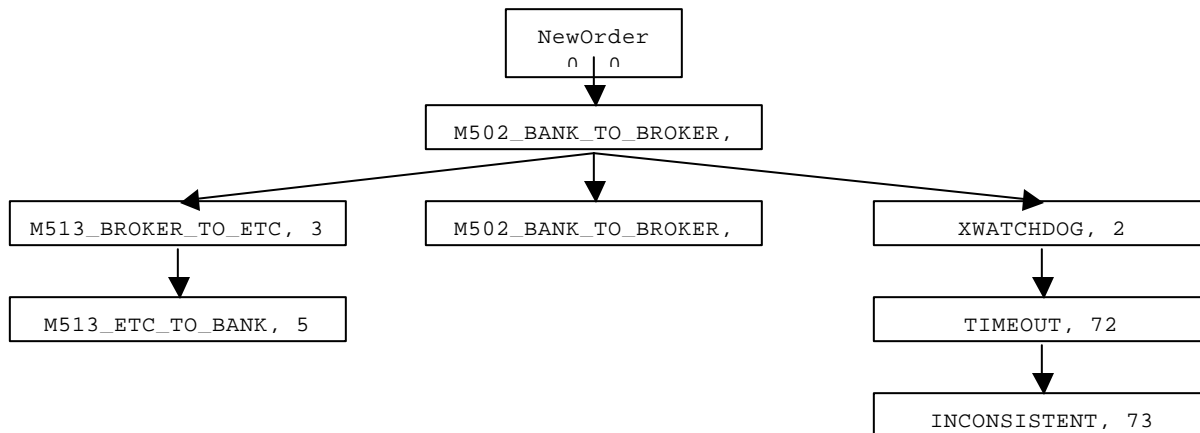


**Figure 5: Causal history of a transaction violating global timing constraint**

Broker's collaborative process interface. Indeed, further scenarios and analysis (omitted here) show that internal business policies of the Broker rejected one of the MT514 events. This means that the partners must revise the collaboration process to deal in more detail with some of the broker's internal business policies. Obviously, it is desirable to detect such situations before the collaboration is put into operation.

### 3. Transaction failed due to race conditions

The poset in Figure 6 shows a race condition in the collaboration protocol. First the bank issues an MT502 order, and then an MT502 cancellation of that order. The poset shows a race condition between the bank issuing the MT502 cancellation and the broker issuing an MT513 confirmation of the order. The two events signify causally independent activities as shown by the separate branches in figure 6. The MT513 confirmation must go to the ETC before it goes to the bank. While the MT513 is en route, the bank cancels. For illustration, we have included symbolic timestamps in the



**Figure 6: causal history of a race between MT502 cancellation and MT513 confirmation events**

events in Figure 6. The MT513 confirmation is issued by the broker at time 3, the bank issues the MT502 cancellation at time 4 and the MT513 is passed on from the ETC to the bank at time 5. Activity stalls because the collaboration has reached an exceptional situation. Finally the global time constraint violates with a Timeout.

This is a race condition inherent in the agreement between the two parties because they are not constrained to synchronize every step they take with the other party. The activities of confirmation and cancellation are executed in parallel, as Figure 6 shows. Analysis of the causally ordered poset has allowed us to recognize this race condition.

#### STEP 6: REFINING THE PROCESS DESIGN

Scenario 2 has uncovered the need for a simple refinement of the collaboration process architecture. The Broker's internal

business policy that failed to handle the MT514 can be expressed as a constraint local to the Broker's interface. This would express that all MT514 messages must meet a specified test. If this local constraint is agreed to as part of the collaboration, and put in place, then the local constraint will be violated before the Timeout. The local constraint violation gives the partners an opportunity to decide how to proceed before Timeout.

The process architecture can also be refined to deal with the race condition in scenario 3. Typically this involves a more complex architectural change. One approach is to add a new message (MT 509) that is sent from the Broker to the Bank and conveys the status of the cancel request. There is a new connector for transporting MT509 events, and new business rules in each of the partner's interfaces for handling them. The refined process model will handle the exceptional cases caused by order cancellations, and indeed, also caused by changes to orders in progress.

The new refined process architecture can be simulated again as indicated in Figure 1. This iterative process of simulation and refinement allows us to subject proposed business collaboration agreements to rigorous testing and analysis.

#### V. CONCLUSION

Complex event processing techniques as exemplified in *Rapide*, present a powerful approach to business process development for the eBusiness enterprise. These techniques can be implemented as supporting tools for many process modeling languages ranging from simple Workflow to more sophisticated process languages such as BPML [1], or WSFL [3]. These languages enable early development of executable process architectures embodying business policies and rules. The use of causal event simulators allows simulation at an early stage in the process development lifecycle, and the analysis of simulation results to improve process design prior to implementation and deployment. Causal event techniques

provide a powerful analytic tool. These techniques based upon detecting complex event patterns and tracking causality between events, have been applied to more complex processes than illustrated here, such as dynamic process architectures encountered in electronic supply chain auctions. A complete discussion of complex event processing is given in [5].

The reactive rules and event pattern constraints used to build *Rapide* process architectures for early lifecycle simulation, can be utilized throughout the later stages of process deployment to ensure conformance to policies and rules during operation. In fact, we propose that monitoring conformance to constraints expressing crucial business policies should always be part of the eBusiness process. Prototypes of the necessary support tools, such as pattern matchers and constraint checkers, have been developed as part of the *Rapide* simulation environment, and can be added to any event-driven middleware used for inter-process communication, such as an ISO-15022 standard financial network.

#### 4. REFERENCES

- [1] Assaf Arkin, A. Agrawal, "Business Process Modeling language (BPML)", Business Process Management Initiative, <http://www.bpmi.org>, Draft 0.4, March 2001.
- [2] Ravi Balakrishnan, "A Service Framework Specification for dynamic e-services interaction," *Proc. IEEE Enterprise Distributed Object Computing Conference 2000*, pp. 28-37.
- [3] Frank Leymann, "Web Services Flow Language (WSFL 1.0)" IBM Software Group report, May 2001, <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>.
- [4] Erik Christensen, et al, "Web Services Description Language (WSDL) 1.1", W3C Note 15, <http://www.w3.org/TR/wsdl>.
- [5] David Luckham, "The Power of Events, An Introduction to Complex Event Processing in Distributed Enterprise Systems", Addison-Wesley Press, April 2002.
- [6] David C. Luckham, John Kenney, Larry Augustin, James Vera, Doug Bryan, and Walter Mann, "Specification and Analysis of System Architecture using Rapide", *IEEE Transactions on Software Engineering*, 21(4): 336 – 355, April 1995.
- [7] David C. Luckham and James Vera, "An Event-Based Architecture Definition Language", *IEEE Transactions on Software Engineering*, 21(9): 714 – 727, September 1995.
- [8] David C. Luckham James Vera, Doug Bryan, Larry Augustin, and Frank Belz, "Partial Orderings of Events and their Applications to Prototyping Concurrent, Timed Systems", *Journal of systems and Software*, 21(3): 253 – 265, June 1993.
- [9] David C. Luckham, James Vera and Sigurd Meldal, "Key Concepts in Architecture Definition Languages", in *Foundations of Component Based Systems*, edited by Leavens and Sitaraman, Cambridge University Press, Feb. 2000.
- [10] James Vera, Louis Perrochon and David Luckham, "Event-based Execution Architectures for Dynamic Software Systems", in *Proceedings First Working IFIP Conference on Software Architecture*, (WICSAI), Edited by Patrick Donohue, pages 303-317, Kluwer Academic Press, Feb 1999.
- [11] <http://www.iso15022.org/>