Conversational Browser

Santhosh Kumaran, Prabir Nandi,
James Hanson, Terry Heath
IBM T. J. Watson Research Center
Route 134
Yorktown Heights, NY
1-914-945-1469
sbk@us.ibm.com

Yashodhar Patnaik
IBM Software Group
Route 100
Somers, NY
1-914-766-1165
pattu@us.ibm.com

ABSTRACT
We present a new programming model for browser-based e-business. This programming model is based on the observation that a browser is used for not just browsing anymore. It is an important part of the internet infrastructure that serves as a key enabler of e-business. We use the notion of conversations to convert the passive browser to an active participant in e-business transactions. The conversational browser is configured using a set of downloadable policies. Conversation policies are used for the sequencing of messages exchanged between the browser and the backend systems. Presentation policies are used to render the views on the browser to create a flexible user interface. In addition to radically reducing the amount of programming needed to create and maintain e-business applications, the new programming model empowers the users by making it possible for them to influence the way the e-business transactions are structured.

1. INTRODUCTION
When the Web browser emerged as the user interface of the World Wide Web, its role was primarily to browse the documents scattered over the internet. As e-commerce and e-business became commonplace, the browser became the thin client that facilitated business transactions on the Web. But the fundamental architecture of the browser did not change. The document browser was adapted to be the GUI that supports a transactional e-business system by ad-hoc additions to the programming model. These additions include browser-based scripting languages such as JavaScript. But the browser remains a passive portal that renders a document stored at some server.

This paper introduces a new programming model for browser-based e-business transactions. The programming model is based on the concept of conversations. A conversation is a sequence of messages exchanged between a set of parties on some topic. An e-business transaction may me modeled as one or more conversations. In our approach, the browser becomes an active participant in such a conversation. A browser is said to be a Conversational Browser when it possesses the ability to participate in conversations.

This paper is organized as follows. We begin with a discussion on conversations as a model for integration. Next, we introduce two examples of browser-based e-business transactions to illustrate the model in action. We present the architecture of an e-business system that supports conversations. We discuss the implementation details of a prototypical system. In particular, we focus on the design and implementation of the conversational browser. We conclude with the analysis of our work and a survey of related work in this area.

2. CONVERSATIONAL MODEL
In the conversational model of component interactions, components (applications, e-businesses, agents) are treated as autonomous, loosely-coupled entities which interact by exchanging messages in a conversational context.

Figure 1 shows the conversational model of component interaction in which the actual component logic is fronted by interoperability technology specifically devoted to managing interactions. The interoperability technology consists of two distinct parts: messaging and conversation support. Messaging is the bare "plumbing" needed to send and receive electronic communications with others. Conversation support governs the formatting of messages that are to be sent, the parsing of messages that have been received, and the sequencing and timing constraints on exchanges of multiple, correlated messages. Conversation support is a separate subsystem that mediates between the messaging system and the component.

This model supports "conversation-centric" interactions between components. This means that messages are sent within an explicit conversational context that is set up on first contact, maintained for the duration of the conversation, and torn down at the end. Each new message in a conversation is interpreted in relation to the messages previously exchanged in that conversation.

Previous work in conversational support focused on business process integration [1] and agent-mediated e-commerce [2]. In business process integration, interaction between businesses (B2B) or within an enterprise (EAI) may be modeled as conversations. Such interactions are carried out typically between servers and are balanced conversations in the sense that all parties participating in these conversations are peers.

In this paper, we extend the conversational model of component interaction to include client-server communications. In particular, we model the interactions between the browser and multiple servers in the context of e-business transactions as conversations. These conversations are unbalanced, since the client and the server are not peers and the conversation pattern is not symmetric. For example, only the client can trigger the conversation.

By providing conversation support in the browser, we propose a new way of doing e-business on the Web in which the end users play a much bigger role in shaping their interactions with the service providers. The scenarios described in the next section illustrate this concept.

3. SCENARIOS

We describe two scenarios to illustrate the use of conversations for browser-based e-business. The first scenario involves a two-party conversation in which a homeowner is trying to refinance his loan. The second scenario is a multi-party conversation in which a traveler uses the browser to make a trip reservation by connecting to airline and hotel companies.

3.1 Mortgage Refinance Scenario

A customer refinances his mortgage loan by connecting to the Web site of the financial institution and providing the necessary information. The server at the financial institution is in control and presents the user with a series of forms. The information provided by the user in these forms is critical not only for the decision making process, but also to determine the specific sequence of forms to be presented. Today, these forms, the processing of information obtained via these forms, and the sequencing of these forms are all programmed at the server side of the interaction, typically by the IT staff of the financial institution. Each financial institution may provide its own implementation of this service and may attempt to differentiate itself from the rest by the look and feel of their site. Nevertheless, the user does not have much of a say in shaping these interactions. He is merely a pawn in the hands of the businesses that offer the service.

3.2 Travel Reservation Scenario

In completing a trip reservation on the Web, a traveler typically connects to multiple service providers including airline companies, car rental companies, and hotels. Today, the user will need to make multiple connections, make reservations separately, and manually reconcile the end-to-end itinerary. We can model this process as a multi-party conversation. This leads to the modeling of total user experience in the context of complex e-business transactions that span multiple service providers.

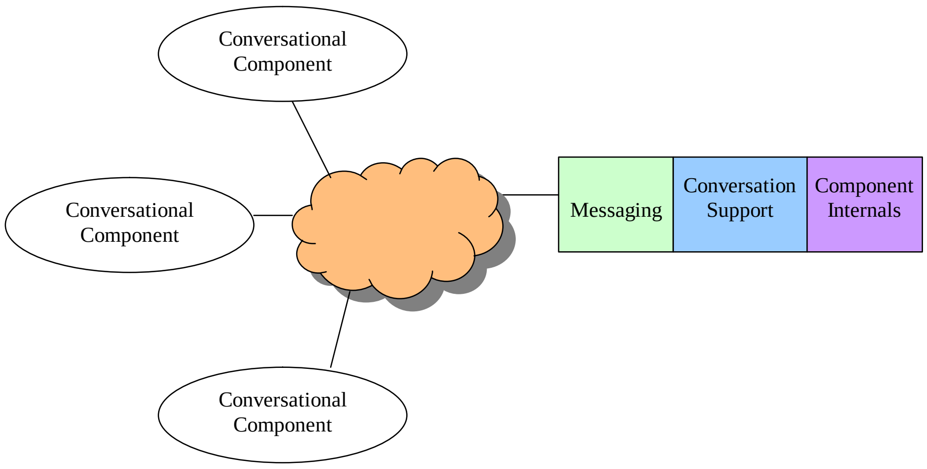

Figure 1: Conversational Model of Component Interaction

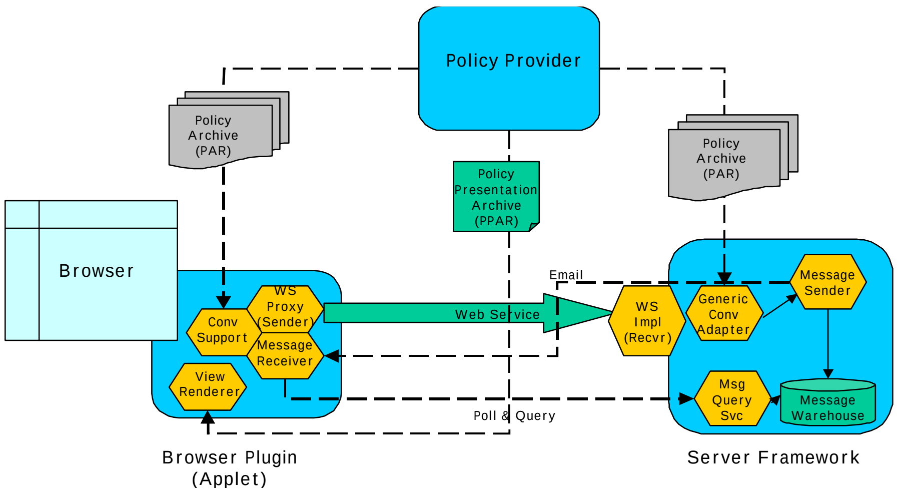

Figure 2: System Architecture

4. SYSTEM ARCHITECTURE

This section discusses the system architecture that supports the conversational browser. In addition to the services consumers and service providers in traditional e-business systems, the conversation model introduces a new role player: Policy Provider. We discuss the role of the policy provider and the facilities required at the consumer side as well as the service provider side to support conversation-based e-business. Figure 2 shows the architecture of an e-business system based on the conversational model.

4.1 Conversation Policies

As illustrated in the section on scenarios, e-business interactions will make frequent use of pre-programmed patterns. Pre-programmed interaction patterns are called conversation policies (CPs). They are the heart and soul of conversation support.

A conversation policy is a machine-readable specification of a pattern of message exchange in a conversation. CPs consists of message schema, sequencing, and timing information. They are conveniently described by a state machine, in which the sending of a message (in either direction) is a transition from one conversational state to another.

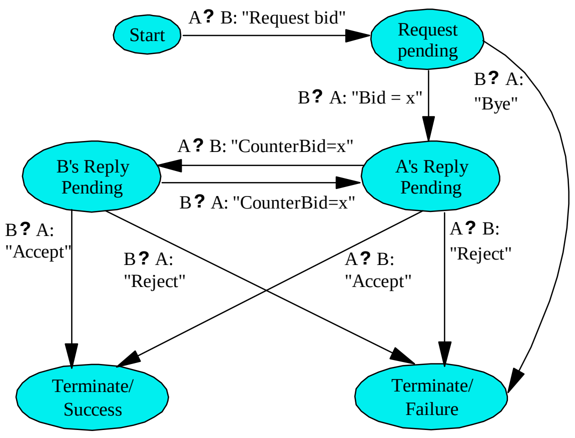

Figure 3: Schematic of a simple CP

Figure 3 shows a schematic of a simple CP, in which two participants, A and B, trade bids and counter-bids until one or the other of them accepts the current bid or gives up. Nodes in the graph correspond to different states of the conversational protocol. In effect, each node represents a summary of what has transpired so far in the conversation. Edges connecting nodes represent transitions from one state to another. Each transition corresponds to a message being sent by one or the other party, and specifies the format or schema of the message as well as which party is the sender. For example, in the starting state (labeled "Start") there is one transition, labeled "A? B: Request Bid", which corresponds to A sending a message to B of the form "Request bid". (Obviously in a real CP, the format of this message would be spelled out.) The CP does not define any other way for the conversation to proceed from its starting state. Similarly, there are two transitions out of the state labeled "Request Pending": one in which party B sends a message to party A of the form "Bid = x" (where x represents some value determined by B), and another in which party B sends "Bye".

In carrying on a conversation, each party separately loads its own copy of the CP, separately maintains its own internal record of the conversation's "current state", and uses the CP to update that state whenever it sends or receives a message. From the point of view of either party, this CP has two types of transitions: transitions to take when a message of a particular format is received, or transitions to take in order to send a message of a particular format. The sender of a message usually (though not always) has to make a decision as to which of the possible alternative messages to send, and often supply data as well--e.g., the value to fill in for bid's amount. Similarly, the recipient usually has to classify the message--identifying which of the possible alternatives was sent--and often parse it to unpack the data supplied by the sender.

As written, the CP is independent of the "point of view" of the company: i.e., which role, A or B, a given company is playing in a given conversation. One can just as easily describe the CP from within one role. For example, if we adopt role A, then transitions labeled "A? B" are interpreted as "send message"; and "B? A" is interpreted as "receive message".

This example is given only as an illustration. In a real system, the message schemas would be spelled out in detail, e.g. via reference to an XML Schema document. And, as we will see below, the overall CP could be broken up into "patterns" or "stanzas", each of which was represented by its own CP state machine.

Some other features to note:

- CPs enable extensive reuse of messages. Because a message is interpreted with respect to the conversation's current state, the same message can be safely reused in multiple contexts. For example, the message "OK" can be used in a bid/counter bid CP to signify acceptance of a bid, in an RFQ CP to signify acceptance of a quote, and so forth. In all cases, the contextual information supplied by the CP and the conversation's current state removes any ambiguity.
- CPs provide economy of expression. No need to make messages self-describing "kitchen sinks" containing all possible context you can think of or might ever want to use.
- Because each of the conversing parties maintains its own record of the conversation's state, and uses its own CPs to update that record, the parties need not, in fact, be using exactly the same CP. The minimal requirement is that, in the course of a particular conversation, the sequence of messages they exchange corresponds, on each side, to some path through the particular CP that party is using.

4.2 Nesting and composition of conversation policies

In day-to-day business, a firm's interactions with other firms tend to be made up of common, conventional interaction patterns. That is to say, its conversations tend to have phases or “stanzas” which fall into common patterns, and are reused in different contexts. For example, first there might be discussion of product discovery, then negotiation of the deal, finally settlement. And it is nested: Product discovery, for example, might start with the customer expressing needs, the seller asking pointed questions about them and then recommending a list of possible matches, followed by the buyer making a selection from the list. Negotiation might start with a discussion of the way to negotiate: haggle over price, or place bids in an auction, or etc., followed by, in both cases, a pattern of message exchange appropriate to that negotiation method. After the products are dealt with, then the parties might turn to a dialog about delivery options (if the goods are physical) and prices. Similar, settlement might start with an enquiry into the methods of payment supported, followed a selection of one of them.

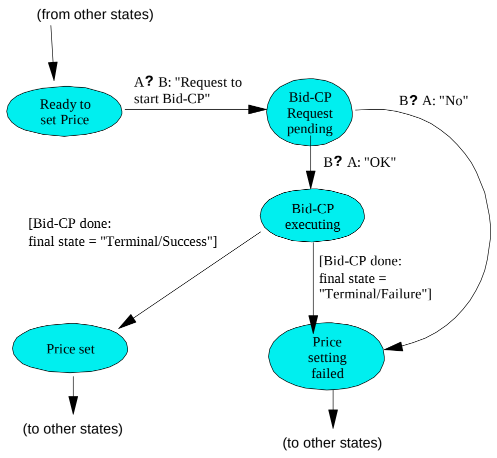

Figure 4: Nested CPs

Conversation policies are inherently nestable. This means that, as part of carrying on a conversation that obeys a given policy, the conversing parties might choose to start a new conversation policy as a “sub-conversation”, possibly carry it out to completion, and then return to the previous conversation policy. In effect, both parties carry on a narrowly-scoped “child” conversation within the enclosing context of the more broadly-scoped “parent” conversation.

For example, two parties might be engaged in a simple negotiated bidding procedure, in which they first identify a set of services to be performed, then they engage in an iterated bidding procedure to settle on a price. That iterated bidding procedure may be represented by the CP in Figure 3, in which the specification of the goods outside the scope of the CP--it is part of the context--and the messages only pertain to the bid price.

In this case, the CP governing the “parent” conversation would contain transitions for starting up a sub-conversation following the bid/counter bid CP. This is shown in Figure 4.

4.3 Policy Providers

In the new business model, we envision an ecosystem of third party policy providers that craft and sell conversation policies. These policies may be created for various industry segments by vendors with expertise in the corresponding domain. Policy providers will disseminate CPs via network-accessible Policy Archives.

4.4 Presentation Support

In Figure 1, the structure of a conversation-enabled component is shown to consist of a messaging layer, a conversation support layer, and an internal layer specific to the component. These layers are seamlessly integrated such that the conversational ability becomes a natural extension of the functions provided by the component.

When the component being integrated is a browser, we are actually extending the presentation capabilities of the browser to create a conversational GUI client.

In a conversational model, when the browser receives a message from a server, it should present an appropriate view of the message to the user, collect the response from the user, and send the response back to the server. It should enable the user to send an unsolicited message to the server by presenting the appropriate form for data collection and form submission. The conversational support mechanisms described earlier facilitates message choreography. But that needs to be augmented with presentation support as well so that the browser can be fully integrated with the conversation layer.

This is accomplished by associating presentation policies with conversation policies. Presentation and conversation are orthogonal functions and the composition of these policies result in the complete script that turns a passive browser to an active participant in an e-business transaction.

A presentation policy associated with a particular conversation policy specifies the views for every message received by the browser in that conversation. Additionally, it specifies the data entry forms needed to initiate messages from the browser to the servers. The presentation policy determines the look-and-feel of the GUI. A different look-and-feel can be obtained for the GUI by associating a different presentation policy to the conversation policy.

We expect third party Policy Providers to distribute Presentation Policies as well. Presentation policy archives serve as the repository of presentation policies.

4.5 Browser Plug-in

A browser plug-in Applet provides conversation support in a browser. The following modules make up this plug-in:

✍ Conversation Support module to support conversations.

✍ View Renderer to support presentation.

✍ Message Receiver to receive messages.

✍ Message Sender to send messages.

Details on the implementation of these modules are provided in the next section.

4.6 Server Side Support

Server side components that participate in conversations need to be integrated with the following modules:

✍ Conversational adapter (CA). This is a JCA (Java Connector Architecture)-based component that provide conversation support for J2EE components [8]. The CA will be configured at runtime with the *Message Sender*, appropriate to the peer's *Message Receiver.* The CA itself will expose a Web Service *Message Receiver* for peer access.

- Message warehouse. This is an optional component and used to support a *store and forward* message delivery with the peer.
- Message query service. This is an optional component in conjunction with the message warehouse to allow peers to poll and retrieve messages from the warehouse.

Details on conversation support modules for the server side can be found in [8].

5. IMPLEMENTATION DETAILS

This section discusses the details of a prototypical implementation of the browser plug-in.

5.1 Conversation Support Module

Conversation Support module provides a light weight implementation of the conversation support function. It is configured dynamically via downloadable conversation policies. A Message Receiver and a Message Sender are associated with it to provide the transport mechanisms for sending and receiving messages. The Applet uses this module to initiate and conduct a conversation session with a compatible server-side component.

The key functionalities of the module are as follows:

- It enables the user to initiate conversation sessions with compatible service providers.
- It provides an interface for the user to send messages in conversation. The messages are exchanged in XML format.
- While in conversation it maintains the conversation state and parses the policy definition to provide the browser (if required) with the type (schema) of messages that can be sent in the current state.
- It provides the interface to register conversation subscribers. Using this interface, a client can register itself as a conversation subscriber. The messages received from the server are delivered to the subscriber.

The Applet instantiates the module and then registers itself as a subscriber to receive messages from the server.

5.2 View Renderer

The View Renderer (VR) module is responsible for providing the presentation support for the conversational browser. The VR module consists of the following:

- Communication Applet: An invisible applet to exchange data between the conversation support and the screens rendered to the user.
- JavaScript Controller: Java script library that handles the client side screen flows, two-way communication between forms, and the communication applet.
- Policy Archive: Policy Archive is a jar formatted file that contains the conversation policy files and the related message schemas. The Policy Archive has the file extension *par.*
- Presentation Archive: Presentation Archive is a jar formatted file that contains the presentation mappers from the message schemas to the screens. The Presentation Archive has the file extension *ppar.*
- Policy Path: This is a configurable parameter similar to the *classpath* parameter passed to a JVM. It indicates the physical location of policy archives (.par), raw conversation policies (.cpxml), and corresponding message schemas. This corresponds to the *jar* or *class* incase of the JVM classpath. .
- Archive Explorer: UI screen flow to manage the archive repository indicated by the policy path.
- Presentation Controller: Controller classes that handle the communication between the conversation support module and the screens.
- Message Formatters: Translators that format XML messages and message schemas to screens and Form data to XML messages. These formatters use XSL style sheets to perform the translation.

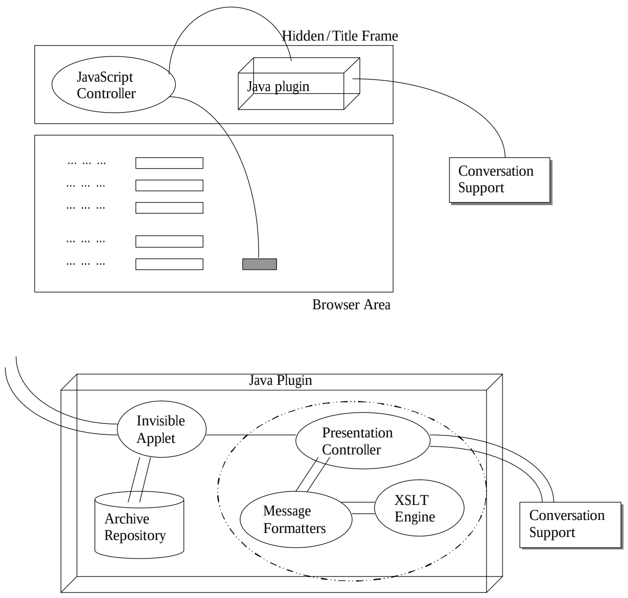

Figure 5: Schematics of View Renderer

5.2.1 Communication Applet and JavaScript Controller

The major functions of these components are as follows:

- Provide a screen flow framework to handle the client side screen flows. The browser uses this framework to create screen flows to explore archive repository and initiate conversation.

- ✍ Download policy archives and policy presentation archives from third party policy providers. A user of a conversational browser can download policy and presentation archives from third party policy providers and extract to the local file system. These components provide the user interface to download the archives. These archives or extracted files thereof are later used to configure and drive conversations.
- ✍ Provides the user interface to manage the policies and message schemas specified by the policy path.
- ✍ Initialize and setup the browser to participate in conversations. The user is required to provide several initialization parameters to initiate a conversation. These parameters include policy name, initial roles, message descriptors, and presentation archives. The browser facilitates the user interface for collecting and initiating a conversation.
- ✍ Facilitate the communication between the markup and the presentation controller for message exchange. Once the user is in a conversation, the conversation proceeds through a series of message exchanges between the participating parties. The messages received are rendered to the user as browser specific markup, and the messages sent from the user are collected and communicated to the conversation support through the presentation controller.

5.2.2 *Presentation Controller*

Presentation controller is responsible for presenting the messages to the user, processing the user messages and formatting the messages through message formatters. These formatted messages are sent to the participating parties through conversation support module. The following are the major functionalities of the presentation controller:

- ✍ Format the incoming messages and provide presentable markup to the plug-in. An incoming message received through a Message Receiver may not be human readable. The presentation controller translates these messages to markup using the "incoming message mappers" from the presentation archive. The translated markup is sent to the applet for rendering on the browser.
- ✍ Retrieve possible actions in the current state of conversation from the conversation support. The presentation controller queries the conversation support for any possible actions in the current conversation state. These actions are mapped to markup and presented to the user. Each action may require a message to be sent, this information is retrieved from the conversation support, and mapped to the markup through the "Schema markup mappers" and/or "Schema translation mappers" from the presentation archive. And the transformed markup is sent to the applet for rendering on the browser.
- ✍ Format the user submitted data to the message schemas. User submits the message data through forms. This data needed to be formatted before handing over to the conversation support. Massage data is collected as name-value pairs from the user. Presentation controller uses the "outgoing message mappers" from presentation archives to format messages to conform to the message schemas.

5.2.3 *Message Formatters*

These formatters are responsible for translating incoming messages and message schemas to presentable markup, and translating outgoing message data to an outgoing message. A message formatter uses message mappers from presentation archive to translate message and schemas. There are four types of mappers:

1. Incoming Message Mapper: Conversing parties in a conversation exchange XML messages. A human participant may receive conversation messages through message receivers. These messages are in XML format. Conversational browser renders these messages to user through mapping to markup screens. A presentation archive contains one mapping per incoming message schema. These are XSL style sheets that translate incoming messages to markup screens.
2. Schema Markup Mapper: As part of a conversation a user sends messages to other conversing parties. When a user is required to send a message, browser renders a form to collect message data. This markup mapper is a screen that renders this form. These screens are defined per outgoing message schema in the presentation archive file. They implicitly assume the structure of the outgoing message.
3. Schema Translation Mapper: Like schema markup mapper, these mapping files map outgoing message schemas to markups that render a form to collect the outgoing message data. These XSL style sheet mappings are more dynamic and can absorb easily any changes in the schema. Unlike schema markup mappers, these mapping files does not "hard code" the screens, the screens are generated by translating the message schema with this schema translator XSL style sheet. A policy presentation archive may provide either a schema markup mapper or a schema translation mapper per outgoing message schema.
4. Outgoing Message Mapper: Outgoing message data is collected from the user as name-value pairs. Outgoing Message Mapprers are XSL style sheets that translate these name-value pairs to outgoing XML messages.

5.3 Message Receivers

Message Receivers implement the mechanisms by which the conversation support receives messages from the server. Since the browser does not support a "call back" mechanism or participate as a messaging endpoint, the following two alternatives are supported:

- ✍ E-Mail Message Receivers: This will be an SMTP client that can download messages from an email server. It is assumed that the server will be sending messages using email as well. The message will be extracted from the incoming mail and an operation (receiveMessageInConversation) invoked on the Conversation support module passing in the incoming message.

- Poll-And-Retrieve Message Receivers: This is the browser side complement to the *store and forward* message delivery mechanism on the server side. This will be instantiated per conversation session and configured with a poll interval. These receivers provide an interface to modify the poll interval dynamically as well. On every poll tick, the receivers query the server for messages pertaining to the conversation (identified by the conversation ID), and if available, retrieve the messages and then notify the conversation support module.

An instance of the conversation support module is configured with a specific type of message receiver. The parameters for the Receivers are passed to the server as "TransportDescriptor" in the "initiateConversation" call to initiate conversations.

5.4 Message Sender

The Conversation Support module on the browser side is configured with a Message Sender compatible with the corresponding Message Receiver on the server side. The Message Sender is essentially a Web Service proxy client [9].

6. MORTGAGE REFINANCING REVISITED

In section 3.1, we discussed a scenario in which a customer refinances his mortgage over the internet. That discussion was based on the existing model of Web-based e-business. In this section, we revisit this scenario and present the implications of the conversational model on the user experience. Figure 6 shows the components participating in this scenario. The modified scenario is as follows:

- The customer begins by downloading conversation policies and compatible presentation policies for mortgage refinancing from an independent policy vendor. Customer selects the policies that appeal to him.
- Customer uses the Conversational Browser to transact with a financial institution for processing the refinance. The Conversational Browser configures itself using the policies, the XML Schema Definitions (XSD) of the messages used by the policies, and the corresponding XSLT transforms specified for view rendering.
- The browser creates a conversation session and connects to the financial institution using Web services to initiate the conversation. A server at the financial institution acknowledges the request, loads the compatible policies, and configures itself for a conversation with the customer.
- From then on, the business process for mortgage refinancing gets executed via a set of messages exchanged between the server at the financial institution and the browser. These messages are choreographed via the conversation policies. On the server side, these messages trigger invocation of back office applications required for the processing of the refinance request. The outcome of these invocations plays a role in determining the response back to the customer. On the client side, the received messages are transformed and rendered as "views" on the browser. The customer responses to these views are converted to messages and sent to the server.

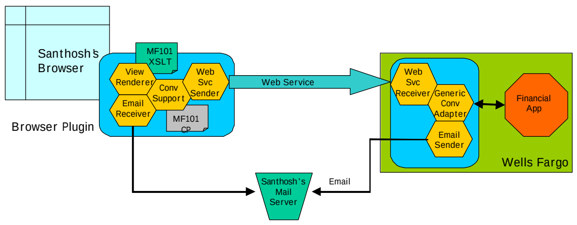

Figure 6: Components for mortgage refinance

7. ANALYSIS

We presented a new programming model for browser-based e-business transactions in this paper. This programming model advances the current technology in several significant aspects as discussed below:

- Empowerment of the end user. As illustrated in the mortgage refinancing scenario, the end users have very limited influence in the manner e-business transactions are structured in the current model. We introduce the concept of conversation policies to define the transactions and the presentation policies to render the views on the browser for user interaction. This changes the model radically, forcing the service providers to support third party conversation policies based on end user preference. If the customer in the scenario has a certain preference with respect to these policies, he probably would prefer to do business with the financial institutions that support these policies. This leads to an ecosystem driven by end user preferences where the end users and the service providers play in equal terms.
- Browsers as Web service clients. The Web services technology is being used today primarily for programmatic integration of server side applications. By making the browser to be a Web service client, we extend the distributed computing model based on Web services to include the desktop.
- From hard-coded programs to policy-driven configurations. In the current programming model, developing an e-business site involves large amount of programming. Developing and maintaining the site is a large part of the operational expenses of an e-business company. When appropriate standards are in place for ubiquitous conversation support, the cost of site development and maintenance could be drastically reduced. Conversation and presentation policies could become a flexible way to customize and maintain an e-business site with very little programming.
- Multi-party transaction. We presented a trip reservation scenario as an example of a multi-party transaction. We

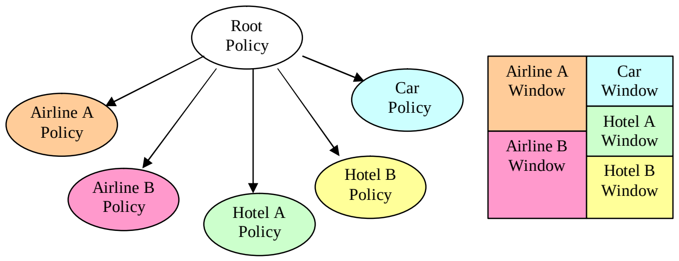

Figure 7: CP Tree for a multi-party transactions

model these transactions as multi-party conversations. For example, in trip reservation example, the user completes a trip reservation by engaging in conversations with multiple service providers, such as airline companies, car rental companies, and hotels. Using the nested conversation policies, we model the total user experience as a dynamic tree with conversation policies as nodes. In Figure 7, on the left side we show a snapshot of the conversation policy tree and on the right side we show the corresponding window set.

8. RELATED WORK

Most of the "conversation" work with Web browsers is in the area of natural language processing and speech recognition techniques embedded in the browser [5]. The conventional 'conversational browser' is essentially a voice-activated browser, providing a mapping between the human voice and text. This could complement the technology we discussed in this paper creating a very powerful user experience.

Marcus Fontura et al propose a peer to peer architecture to create Internet auctions [5]. The auction policies to specify auction algorithms (for example, Open-cry, Dutch etc.), handling of certification, auditioning, treatment of complaints and the actual auction rules itself are defined as 'laws'. The paper illustrates an example of an *auction law* that is used by the auction agents to conduct peer to peer communications. The browser (if employed) continues to be used traditionally, serving up content from the server. These and other such architectures provide some form of decentralized interaction support between two components but do not use the browser as a full participant in the conversation, as proposed in this paper.

There are some browser based products that drive interactions with a user using a decision script [6]. Their applicability is primarily in the area of providing customer support. The script runs on the server and guides the customer through the troubleshooting process.

The Nareau Platform [7] is an internet wide collaboration environment aimed at facilitating conversations and gaining control of one's computing experience via writing, automating, and streamlining tasks. It is based on the notion of "server-enabled" desktops that uses "rules" on incoming and outgoing messages.

Hanson, Nandi, and Kumaran discuss the use of the conversational model for server-to-server business process integration in [1]. Conversations are used as an agent collaboration mechanism in [2]. This paper builds on these ideas and extends it to use conversations as the cornerstone of a new programming model for browser-based e-business transactions.

9. CONCLUSION

The Web browser emerged as a standards-based client for browsing the internet. As internet became the backbone of e-commerce, browser began to play the role of a thin client for e-business transactions. Conversational browser, as discussed in this paper, has the potential to revolutionize the browser-based e-business transaction model in two ways. (1) By enabling the browser to be an active participant in the business process execution, it provides a clean and powerful mechanism to integrate the desktop with the business processes. (2) By supporting the policy-driven customization of the user experience and the message choreography, it simplifies the programming model. But these changes are possible only if standards are in place for ubiquitous conversations.

10. REFERENCES

[1] J. Hanson, P. Nandi, S. Kumaran, "Conversation Support for Business Process Integration," Enterprise Distributed Object Computing Conference, EDOC 2002.

[2] J. Hanson, P. Nandi and D. Levine, "Conversation-enabled Web Services for Agents and E-business", Proceedings of the Third International Conference on Internet Computing (IC-2002).

[3] Conversation Support for Web services, http://www.alphaworks.ibm.com/tech/cs-ws

[4] The W3C Voice Browser Workshop (http://www.w3.org/Voice/1998/Workshop/papers.html)

[5] Law-Governed Peer-to-Peer Auctions, Marcus Fontoura, Mihail Ionescu, Naftaly Minsky, IBM Almaden Research Center, Department of Computer Science Rutgers University (http://www2002.org/CDROM/refereed/398/)

[6] Smart Servers, Vanguard Corporation (http://www.vanguardsw.com/DecisionScript/SmartServers.htm)

[7] The Nareau Platform (http://nareau.weblogs.com/)

[8] J2EE Connector Architecture 1.0 Specifications (http://java.sun.com/j2ee/connector/)

[9] Web Services, (http://alphaworks.ibm.com/webservices)