Chapter 1

UTILIZING WEB SERVICES IN AN AGENT BASED TRANSACTION MODEL

Tao Jin & Steve Goschnick
Department of Information Systems, University of Melbourne

Abstract: The web services paradigm is an opportunity for a universal programmatic interface to the Internet, one that could parallel the web-browser, human user-centered interface, in scope and adoption. However, there remains a great challenge in being able to perform reliably transactions in the loosely coupled web services environment, hosted by independent web service providers, running on heterogeneous systems. The ACID properties in traditional transaction models cannot be guaranteed in this environment, due to unreliable communication, the uncertain duration of individual services and the decentralized nature of server management. We draw lessons from the long history of research into distributed transactional systems. We note that the recent WS-Coordinate protocol, draws upon some but not all of those lessons. Unlike WS-Coordinate, we are interested in being able to incorporate independent take-it or leave-it web services, into orchestrated systems - not just those that agree at the outset, to provide a particular service, with a mutually agreed quality-of-service regime. To address these issues, we call upon the flexibility of the agent-oriented paradigm, and propose an agent-based transactional model (ABT) that orchestrates multiple, often independent web services, into new robust services. We assert the use of multi-agent technology to manage and orchestrate transactions, is the right choice in the web services environment - the choice most-likely to propel the web services paradigm to levels of adoption that approach those of the web-browser interface.

Key words: web services, transaction model, agent architecture, multi-agent system (MAS)

[**This publication**: Tao Jin & Steve Goschnick (2003) Utilizing Web Services in an Agent-based Transaction Model (ABT). Proceedings, Workshop on Web Services and Agent-based Engineering, at the AAMAS-2003 conference, Melbourne, Australia.]

1. INTRODUCTION

Where HTML and the web-browser together represent a universal *human* interface to the Internet, web services offers us the opportunity of a universal *programmatic* interface to the Internet. Web services offer a new solution for dynamic, automated interactions over the Internet, made possible through appropriate standards and adhering technology. In recent years, many major IT industry players have ventured into this area, staking a claim for their respective futures. At the grass-roots level too, the use of web services is expanding, driven by the growth of new devices, new service opportunities, application-to-application communication, system interoperability and a new mechanism to expose pre-existing services programmatically and openly via the Internet.

W3C has defined a web service as *a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols* [9]. Depending on who you ask, web services are either an incarnation, a generalization, or a reinvention of distributed computing [1]. However, unlike "*tightly coupled*" traditional distributed systems, the web services environment is loosely coupled. So far, organizations provide and promote their web services independently; however, multiple web services can be orchestrated in a cooperative manner, to form a more dynamic and complex service, often unforeseen by the individual web service providers. Much research has been conducted either from a business model's perspective such as web aggregation [31] or from an agent's perspective such as synthesis and enacting of web services [5, 20]. Whatever it is called, this kind of orchestration of disparate services nevertheless gives rise to a significant challenge in conducting *transactions* (or something very much like them), that incorporate disparate, independently hosted web services – as we will now outline.

A transaction is *a collection of operations on the physical and abstract application state* [22]. In practice a transaction is enacted like so: `BEGIN: task 1; task 2; … task n; COMMIT`. Conventional transaction processing conforms to the so-called ACID properties [24] as follows: Transactions must guarantee that either all operations are completed or otherwise, none of them are enacted (*Atomicity).* A successful transaction commits only legal results, from one consistent state to another consistent state *(Consistency)*. Events within a transaction must be hidden from other transactions running concurrently (*Isolation).* Once a transaction has completed and has committed its results, the system must guarantee that

these results and changes to the state survive any subsequent malfunction or failures *(Durability)*. To strictly conform to all four ACID properties, the transactions are necessarily short-lived. It also requires a closely coupled system where a *transaction coordinator* has control over all the transaction participants, since participants in a transaction must effectively trust that the transaction coordinator will not lock their resources indefinitely. In the early 1980's a number of distributed transaction models were proposed and widely adopted [3, 23, 24, 32]. These models require full central or third-party authority to control sub-transaction participants and resources. This is necessary for controlling the status of each participant and for appropriate handling of resource-locking issues.

However, the web services environment currently lacks any such level of authorization. Organizations that provide individual web services are unlikely to give up control to a remote, unknown or untrusted party. Any unusually long interactions (latency) with other concurrent activities or with a web service transaction of uncertain duration, will often break the *atomicity* boundaries as seen in traditional transaction processing. Furthermore, the *isolation* property becomes an issue due to potentially indefinite locking of resources, and so the providers of individual services are unlikely to be willing to let others lock their resources at all. Additionally, Internet communication is much less reliable than that of an Intranet. Therefore, the traditional *abort-restart* or *compensation processes* approach to achieving reliability (as proposed by the WS-Coordination approach below) become very expensive due to frequent failures and an unpredictable quality of service.

To address many of these issues, several standards are underway which attempt to get web service providers to agree at the outset, to certain constraints to aid web orchestration in a contract-oriented manner. One such effort is the WS-Coordination protocol specification [17, 30], and another is the W3C Web Services Choreography Working Group, who are currently gathering requirements for a web services choreography description language (CDL), which they themselves describe as a work-in-progress [42]. In the WS-Coordination approach, web service coordination requires all parties to agree to a third party having a coordinating role, and that all participants that perform an atomic transaction, are able to perform a compensating transaction, to undo an earlier atomic transaction - when requested to do so by the coordinator.

This web service coordination approach requires web service providers to know in advance, in what way their services will be orchestrated with others, and to agree to conditions in its use within them. Our model outlined in this paper, requires no such pre-conditions on individual web services. To comprehend how much the web service coordination approach would

impede the adoption of web service orchestration as a distributed computing paradigm, when compared to our model, we offer this thought. Try to imagine in the past, if such a condition had been placed on a web-page design before a web browser would show it, how much less of a phenomenon would the *HTML-webbrowser-www* Internet have been, then we currently know it?

The unusually long interactions issue (latency) has been actively researched in the area of workflow management, open-ended activities, and internet applications in recent years and a great number of advanced transaction models have been proposed [4, 8, 11-13, 16, 26, 28, 29, 34-36]. Although these models were not designed to work in a web services environment, they nevertheless provide a solid conceptual foundation upon which we develop our model.

Our research is premised on the view that work done in transaction processing, particularly that based upon distributed systems, is likely to provide insight and direction, as we go about orchestrating web services. Furthermore, we saw an opportunity to use agent technology to overcome the failure of a web service (outright failure, or a time-based perceived failure), by appropriately substituting another like-functioned service, from within a cohort of such services. By doing so, we negate the need for a prior agreement on cooperation. Taking up that opportunity, this paper outlines a new agent-based transactional model, *ABT*, applied in the distributed web services environment.

In building a new orchestrated service, we engineer a number of web services as agents performing the role of *transaction coordinators*, and we wrap external web services as *resource provider* agents. Additionally, we are able to marshal local resources as part of the system. These agents together form a dynamic multi-agent system (MAS) and work cooperatively in orchestrating web services in a reliable and robust manner – without the need for a preexisting agreement to do so. The rest of this paper is organized as follows: In section 2 we briefly review related research areas. Our main focus is on transactional models and agent technology. Then we present our transaction model in section 3. Finally in section 4 we conclude our work and propose further research.

2. RELATED WORK

2.1 Transaction Models

One stream of advanced transaction model research is based on task/workflow modeling. Some earlier models include the notion of a task and flow of control between tasks is specified by an augmented *Petri Net* model [2, 41]. Tasks are represented by position nodes and flows between tasks by transition nodes. When a task completes, it marks its corresponding position node and the transition may fire. *Trigger* model [10] offers data or event driven specification of control flow. Triggers are condition-action pairs. When condition becomes true, trigger is fired and tasks are executed. Trigger model provides a flexible and modular framework with which the control structures of the activities can be extended or modified. *Activity transaction model* (ATM) [11] has a model for long-running dynamic workflows and each workflow uses a trigger to specify and organize long-running activities. ATM supports extended recovery options of roll back and compensation as well as forward recovery. The *CovaTM* model [26] treats a transaction as one execution of a cooperative process with its sub-transactions corresponding to activities. It introduces user intervention to enhance the system flexibility and ability. It also incorporates time predicate to depict time-critical and none time-critical activities uniformly. An activity or sub-transaction in CovaTM may be reactivated after its submission.

Another stream of research is focused on structural engineering. *Nested transaction* model [32] extends the flat transaction structure to a multi-level one. Instead of recording a savepoint after some pieces of work, nested transaction puts a lower level sub transaction which can be committed or rolled back individually. A nested transaction is a tree of transactions, the sub-trees of which are either flat transactions or nested transactions. If the parent transaction is aborted, all its children are aborted. Isolation is guaranteed at a global level as the committed data is released only after all the parent transactions are successfully completed. The *Split/join transaction* model [34] allows a user to commit changes to some of the data objects that will not change, while the remainder of the transaction continues to operate on other objects. Split transaction divides an ongoing transaction into two serializable transactions and then each sub transaction may proceed independently with its own data. Finally, a join transaction, the inverse operation of split transaction, can combine results together and release them atomically. The *Transaction groups* model [13] provides an extended set of non-restrictive and communication (notification) modes to facilitate sharing of data among collaborators. The *Flexible Transaction* model, as well as the

mixed transaction model [12], is designed for multi-database environments and allows more flexibility in transaction scheduling and permits functional replication of transactions. The composition of flexible transactions can tolerate failures of individual sub transactions by taking advantage of multiple database system (In a sense, this foreshadows our approach in using like-functioned web services as alternatives - rather than duplicated services). The mixed transaction allows compensable and non-compensable sub transactions to coexist within a single global transaction.

Long Lived Transactions have been actively researched in the past few decades (The WS-Coordination specification, effectively equate *Long Lived Transactions* to *Business Transactions*). The notion of *saga* [16] is proposed in 1987 as a base model for long-running activities. The Saga model defines a chain of transactions as a unit of control and relaxes the requirement of the entire transaction as an atomic action by releasing some resource before they complete without sacrificing the consistency of the database. It requires a compensating transaction within every sub transaction to provide a backward recovery process - this is precisely the mechanism proposed by WS-Coordination. The Atomic property of ACID is achieved by starting the compensation sequence whenever the transaction that is currently executing rolls back. The system does not roll back a sub transaction in the case of crash, instead, it runs the chain of compensation transactions backwards [22]. Saga has some limitations such as the linear workflow sequence and the expensive compensation policy.

Other extended transaction models were developed later to overcome the limitation of saga by supporting more general and more powerful sets of control flow descriptions and permitting those descriptions to evolve over time. The *Migrating transaction* model [29] extends the saga model by permitting concurrent execution of the atomic transactions. The linear sequence in saga can be extended to an acyclic graph. The *ConTracts* model [35, 36] even extends migrating transactions. It identifies the issues of maintenance of context information and *forward recovery* - from a previously stored good *state* of the distributed transaction, one reapplies the transaction up to and beyond the point of breakage. The steps of a contract may be sequential programs, not just transactions. The execution of each step and the control flow among the steps are specified as part of the contract definition. The *ASSET* model [4] consists of a set of transaction primitives that allow users to define custom transaction semantics to match the needs of specific applications. The transaction primitives can be used to specify a variety of transaction models, including nested transactions, split transactions, and sagas. Application-specific transaction models with relaxed correctness criteria (this foreshadows our use of *Constraints* in ABT to

increase flexibility), and computations involving workflows, can also be specified using the primitives.

2.2 Internet Transaction Models

In recent years, business activity has expanded into internet-based processing and the web services paradigm is being measured as a mechanism to automate some of that processing. All the transaction models mentioned above are designed either for single node or controlled distributed system in an intranet environment. None of them have become widely used in an Internet context. To provide a transaction standard for the Internet, a number of transaction protocols have been proposed. *Transaction Internet Protocol (TIP)* [28] is a very simple two-phase commit protocol employing the two-pipe model, which operates over TCP/IP connections. Compared to other models relevant to internet application, TIP has two innovations. The first is the *pull* method of transaction propagation. A new participant can request an existing participant (superior) to add it to the transaction. The second is the *delegated commit*. The transaction demarcation is controlled by the client application, but only the server node participates in transaction commitment and recovery - this is a coordination task. This is most appropriate for internet applications where the client has no local recoverable resources. Furthermore, an application may choose whichever protocol mechanism which is most appropriate to communicate with a partner.

OASIS proposed *Business Transaction Protocol (BTP)* [8], a specification for an XML based protocol for managing transactions on the internet. Like TIP, BTP also incorporates a two-phase commit protocol to control work flow. BTP is designed to allow coordination of application work between multiple participants owned or controlled by autonomous organizations. BTP relaxes the ACID requirements by introducing the concept of *Atoms* and *Cohesions*. While a BTP atomic transaction still guarantees ACID property, the cohesive transaction allows some participants to confirm and some others to cancel. It also defines some actors and roles as well as their relationships. There are two primary relationships. One is the control relationship between the *Terminator* (the application element that determines the completion of transaction) and the *Decider* (the BTP actor on top of a transaction tree that decides whether the participant confirms or cancels). The other one is the Superior-Inferior relationship within the transaction tree, where the *superior* (coordinator) informs the *inferior* (participant) what the outcome decision is. We believe BTP is on the right track evolving towards internet transaction standards. It addresses the common internet transaction problems such as long duration, resource locking and unreliable communication. However, BTP only specifies a set of

messages between transaction parties and leaves much to do for implementation. It requires every transaction participant to implement a complex work flow and communication protocol. Although we have already seen commercial products that implement BTP such as HP's *Web Services Transactions (HP-WST)* [25], its complex message structure and sophisticated workflow management has so far prohibited its wide acceptance. Moreover, it still lacks of a flexible recovery process. For example, in the case of an atomic transaction, if a certain participant fails, it requires all other participants to roll back. This might be an expensive practice as failures are common in internet environment.

The aforementioned WS-Transaction/WS-Coordination model, like BTP, has the requirement that all players adopt new protocols - on top of WSDL, in their case. WS-Coordination also breaks bit-players into *atomic transactions* and a more complex *business transaction.* Atomic transactions are those which are short-lived, capable of two-stage commit, and capable of rollback via a compensating transaction. Business transactions are those with long life times, sometimes involving human user input (in agent terms, this is called *human-in-the-loop*).

So far we have reviewed many advanced distributed transaction models from the literature. Some of them are developed from a database point of view and some are workflow-centric. They nevertheless give us valuable ideas and knowledge to develop our model. During the process of system development, we found that building transaction participants as web services is very similar to engineering agents, and we could see advantages in the agent oriented approach, over current initiatives such as WS-Coordination - itself based firmly on some of the distributed transaction research outlined above.

2.3 Multi-Agent Frameworks with Hierarchy of Roles

Intelligent software agents, *sense* (inputs) the world they operate in, *deliberate* on how to best achieve their *goals*, then *act* (outputs) upon their world with the *intention* of achieving a goal [38].

Multi-Agent systems are intended to achieve their goals via cooperation between multiple individual agents working in cooperation on jointly held goals. A number of MAS architectures have been proposed [19, 33, 37, 38, 40]. Basically there are two streams of cooperation between agents in a multi-agent system, either via *Teamwork* or by *Collaboration* [7]. Teams are formal and often rigid representations of relationships between individual agents, and could be considered to be analogous to the WS-Coordination approach to web service cooperation. Collaborations between cooperative agents are loosely coupled and have indirect benefits and penalties. They are

more suitable for dynamic, heterogeneous environments – such as the environments encompassing web services.

Achieving cooperation amongst heterogeneous agents requires communication in a common language, or message protocol. A number of high level languages have been proposed for this purpose, including KQML [14] and FIPA [15]. In the web service arena, the mechanism to exchange information between different web services is SOAP. SOAP is a stateless, one-way message exchange paradigm and works across the Internet. Applications can create complex interaction patterns by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information.

We were interested in building our ABT transaction model upon the *Shadowboard Agent Architecture* [18], as depicted in Figure 1. Although it should become apparent to the reader that other MAS agent architectures, particularly those that have a hierarchy-of-roles (a *role* comes with the responsibility and capability of achieving certain tasks), could be equally adapted to suit our purposes. It has an individual *whole agent* made up of numerous sub-components as follows: In the center of the whole agent is the *Aware Ego Agent* – the dominant sub-agent in the whole cluster of sub-agents. The Aware Ego Agent is surrounded by first level sub-agents, which it coordinates with the aid of a built-in constraint logic language (called CoLoG). Sub-agents can also have second level sub-agent clusters of their own. Clusters can be formed recursively to any level as required.

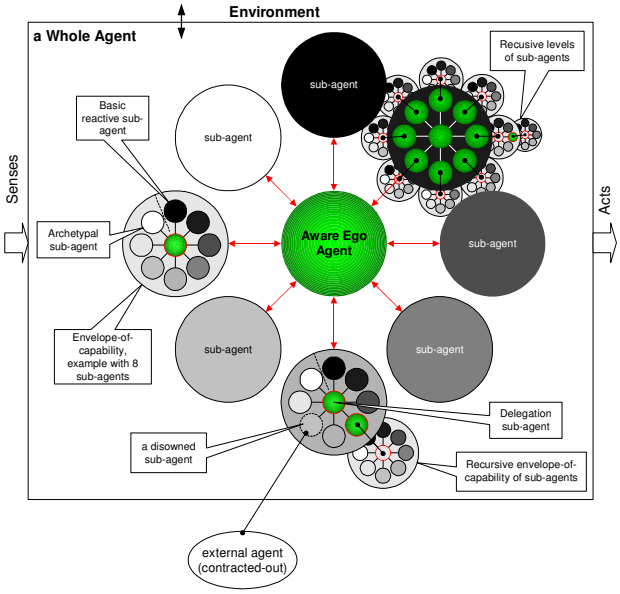

Figure 1. Shadowboard Agent Architecture

The Aware Ego Agent, as well as each of the other delegation sub-agents, have knowledge about the capabilities of sub-agents in their *envelope-of-capability* (a grouping of like-functioned agents). The delegation agent uses this knowledge to select the appropriate sub-agent within its envelope-of-capability to achieve a particular goal, received from higher up the recursive hierarchy - ie. goal-driven processing. When that goal has been achieved - in our case via the services of numerous web services - they pass up the result in the form of a predicate of ground terms - ie. data-driven processing.

When a sub-agent has been found lacking in capability to achieve a specific goal – ie. it has either failed in its task, or it has failed to return a result within a specified timeframe – the delegation agent can select an alternative agent from the envelope-of-capability. By wrapping a web service within a sub-agent, and then orchestrating many such web services as transaction participants, we have been able to build-in high tolerance to failure, without the need for prior conditions between web service providers.

3. THE AGENT-BASED TRANSACTION MODEL

In the web services transaction environment, there are a variety of parties that can participate in a single transaction. Participants can be coordinators, application elements, local/remote resources, internal/external web services, or even orchestrated composite web services. Internal resources (eg. a database-oriented service) in a large enterprise can be on different platforms. In order to simplify the control logic and develop a universal model among heterogeneous participants, we wrap a local resource as a web service, and then use the agent-oriented approach to wrap each web service as a software agent, within our hierarchy of sub-agents, which together make up a MAS. Each agent has the flexibility to carry out its internal operations autonomously and proactively - of course the internal sophistication of each web service will vary greatly. Thus the internal behavior and control logic is isolated from the overall transaction level, allowing the incorporation of existing unrelated web services.

3.1 Role Definition

We define a number of roles that transaction agents undertake within ABT. These roles have their counterparts in traditional distributed transaction standards such as X/Open DTP [39], though the scope of Communication Manager and our WA is slightly different. Table 1 describes their roles, responsibilities and traditional counter-part role respectively.

Table 1: Agent role in ABT

Agent Role	Responsibility	Traditional Transaction Role
Coordination & Delegation Agent (CDA)	*Determines appropriate sub agents and invokes sub agent service*	Transaction Manager
Resource Agent (RA)	*Wraps and manages local resources*	Resource Manager
Wrapping Agent (WA)	*Wraps Web services and relays the coordination*	Communication Manager
Discovery & Rating Agent (DRA)	*Discovers available external services and evaluate their quality*	Nil

The *Coordination & Delegation Agent* (CDA) contains transaction logic to decide which participants to engage in a transaction. In the Shadowboard implementation, the CDA is an envelope-of-capability, and the enacting of logic is done using predicate logic rules and constraints, via the built-in CoLoG language. It accepts requests from the upper level and coordinates the service of lower level agents. It is engineered to form a hierarchical structure by communicating *requests* (*predicates* in the form of *queries*) and *results* (*predicates* in the form of *ground terms*, or *false* on failure) between different hierarchical levels. Middle level CDA agents are able to receive results from further down, and are able to push their combined results up the hierarchy. When it is coming from above it is referred to as *goal-driven* processing, when it is passing results up, it is referred to as *data-driven* processing.

Resource Agent (RA) represents local resources participating in the transaction. Local resources include traditional transaction participants such as tables, databases or application elements. RA is responsible for managing resource locking, state change and any sub transaction control issues. It acts as a resource manager with agent capabilities which effectively isolate resources from CDA. In the Shadowboard implementation, RA is an envelope-of-capability, in which all of the sub-agents are local resources with ranging degrees of sophistication.

Wrapping Agent (WA) wraps web services. Unlike RA, which needs individual programming to work with different kind of resources, WA only works with web services participants. Since web services have standard interface definition (WSDL) and a message exchange protocol (SOAP), WA can be engineered to program itself to wrap existing WSDL services, enabling them to pass transaction messages to the other web services in an orchestration. WA can be used to delegate either single or composite orchestrated web services from the Internet. Some large organizations may have already implemented intranet-based web services. For model simplicity, WA can also be used to wrap these internal web services.

Discovery & Rating Agent (DRA) is attached to CDA to provide supplementary services. It expands CDA's knowledge of existing as well as emerging web services by querying public discovery services such as UDDI. Furthermore, the nature of intelligent agents makes it possible to autonomously evaluate the quality of various web services that have been engaged. CDA monitors results from its participating sub-agents, including their time to respond, and passes these management metrics to DRA. DRA updates their service rating by evaluating the availability, performance, reliability and success rate. By exchanging web service rating between CDA and DRA, CDA is able to improve service quality by orchestrating combinations of better performing web services.

3.2 Rule Definition

We also defined a number of rules to model the multi-agent architecture as in Table 2. The intention of these rules serves for the following requirements. First of all, we would like to take advantage of agent capability to simplify user operation. Agent should autonomously carry out all the activities in a black-boxed manner, without user intervention. Secondly, the hierarchical structure is necessary for complex transaction requirements, and it defines the order of invocation of the underlying logic rules, predicates and constraints - in the case of CoLoG within Shadowboard. The sub CDA interacts with upper level CDA in the same way that Root CDA interacts with user. Thus intervention between different levels is kept to a minimum. Finally, due to different interfaces and behaviors of various resources, we establish a one-to-one relationship between RA and resources. These rules form the basis of the ABT architecture, which will be covered in the following section.

Table 2: Agent rule in ABT

Rule	Description
Rule 1	*Only one agent is visible to user applications. Message exchange between the user and this agent should be as simple as a request/response*
Rule 2	*Only one type of agent can engage multiple sub level agents and coordinate their actions, i.e. CDA*
Rule 3	*Coordination can be spawned and nested. Only one agent is visible to upper level coordinator. Spawn level is unlimited. This is the extension of Rule 1*
Rule 4	*Each resource or web service has only one wrapping agent and a wrapping agent can only wrap one web service (WA) or one resource (RA).*

3.3 The Architecture

The architecture of ABT is based on the roles and rules defined in previous sections. In the centre of the whole transaction agent is the Root CDA (enacted by the Aware-ego agent in the Shadowboard implementation), which is the only contact point to the requesting user application. The Root CDA engages a number of RAs for coordinating resources and WAs for web services. Each CDA has an associated DRA for discovery and rating service. To handle complex transactions, CDA can "clone" itself to form one or many sub structures based on the results of the discovery service. Ie. the hierarchy is mutable, meaning it can grow during the life of the system.

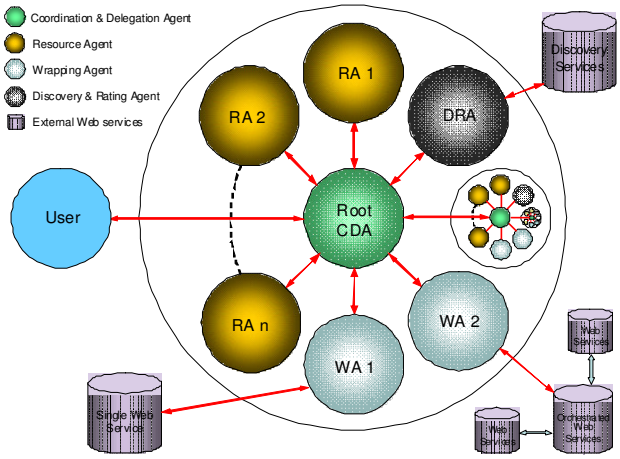

Figure 2. The Architecture of ABT

Each sub CDA also engages a DRA and a number of RAs and WAs to form a complete sub transaction. It regards Root CDA as a user application and the only communication between different levels are through CDAs. Theoretically the number of sub structures and the levels of hierarchy are unlimited. The diagram is depicted in Figure 2.

Conceptually, the ABT architecture is an expansion of the Shadowboard architecture through the introduction of the discovery and rating component, and by defining it in transactional terms, rather than in predicate logic and constraints. All the rules and roles defined in ABT are currently instantiated via the Shadowboard architecture, by modeling the web services as transactions.

3.4 The Message Flow

Our agent models utilizes web services interface, and so naturally the messages exchanged between CDA and their child WA and RA agents are SOAP based [9]. The message types defined in this model are high level though simply stated.

We defined three types of basic messages, *Request*, *Confirm* and *Cancel*, between user and Root CDA (as well as between parent CDA and child CDA). The CDA-to-CDA messages are currently not SOAP-based, they are in an internal format, as all CDA are operating on the same machine – the WAs are distributed. The user application initiates the transaction by sending a Request message, together with *transaction context*.- effectively a call-stack between currently participating CDAs. The Root CDA then autonomously coordinates the whole process without further user intervention. Once the transaction completes, it sends back the Confirm message, or in case of total failure, the Cancel messages.

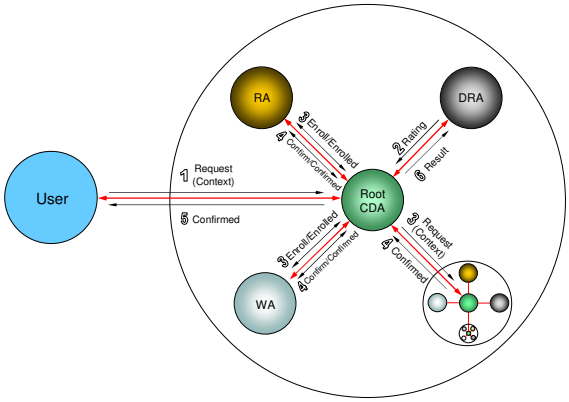

Figure 3. The Message Flow of ABT

We employ a two-phase commit protocol between various agents. This is the tried and trusted protocol for Internet environment. CDA gets the rating message from DRA to select the appropriate participants, then it sends Enroll message to all participating RAs and WAs. In the case of WAs an active response from the web service is taken as an enrolment. Once all RAs and WAs reply with Enrolled message to complete phase 1, CDA then sends Confirm and waits the Confirmed reply. In the case of WAs, a SOAP message with a result, is taken as a Confirmed reply. Ie. A Confirmed message equates to 'Goal achieved' and the appropriate result is passed up the hierarchy. The interaction between CDAs is the one phase approach. Once all participant agents have confirmed their sub transaction, The Root

CDA sends the Confirmed message to the user and the whole transaction is completed. Figure 3 illustrates the message flow between agents in a successful transaction.

3.5 Failure Recovery

Every transaction model should address a recovery strategy in case of failure, either backward recovery or forward recovery. Operations involved in the recovery process include rollback, restart, abort or compensation. The whole transaction will only be successful after all participants succeed. This is essential to achieve atomicity. However, whatever methods are used, they all face a common problem: failure renders previous operation efforts useless, and yet more effort is required to bring back the participants to their original states. In a tightly controlled environment this is not a big problem because usually failure is rare and atomicity is more of a concern. In the web services environment, where the nature of internet is unreliable and no web service can guarantee 100 percent availability regardless of network or server conditions, failures are more common and unpredictable. If certain services fail for an indefinitely time period, traditional transaction will never be completed. If the coordinator chooses to abort the transaction, it is also too expensive to request all successful participants to roll back or compensate. This greatly impacts the quality of the whole system.

ABT adopts a different approach for failure recovery. We call it *alternative participant approach*. The CDA coordinates its participants by transaction rules and has the knowledge of a number of web services that provide similar functionality. Each can act as the others' alternative. Through the continuous input from DRA, CDA expands its knowledge of available services and their quality. Once a transaction is requested, CDA invokes the appropriate participant agent based on their ratings and keeps others as backups. If a participant fails due to whatever reason, CDA abandons it and invokes another similar agent from its backup list. Thus the whole transaction goes on. Other agents either in the same level or further up the hierarchy may not even notice this failure. In most situations the CDA also performs a reporting function, like the use of the call-stack within a language compiler, it can detail the particular web services used and their known qualities via DRA, qualifying the results. This non-stop approach, saves the expensive rollback or compensation of sub transaction modules, seen in other approaches.

In the worst scenario, where all the alternative services for a critical participant fail, our model does allow the whole transaction to abort. The CDA sends Cancel message to its parent and to all participating agents to abort. Sub agents implement their own recovery strategies based on their

transaction model. The possibility of this scenario depends on a CDA's knowledge. The more web services CDA knows of and can call upon the less likely a full abort will happen.

3.6 Discussion

Strictly speaking, our method of choosing alternative participants breaches the atomicity rule of the ACID properties of the traditional transaction. Traditional transaction concept requires either all operations be completed or otherwise none has happened. We argue that such a requirement restricts the development of dynamic Internet transaction models. The fact that alternative WAs are capable of returning different results, simply reflects the pluralistic nature of the global Internet. It is a non-deterministic system, which reflects the world we live in. Other Internet transaction models relaxed the atomicity condition as well. BTP[8], for example, partially relaxes atomicity by introducing the concept of Cohesion. In ABT, an atomic unit is not mapped to individual participant. Instead, we consider each logical service to be an atomic unit. CDA chooses the appropriate service to form a complete single logical service. The control logic makes sure all logical services must succeed, or otherwise all fail. Thus, we can still guarantee atomicity at the overall transaction level.

The benefit of ABT, compared to other traditional models, is obvious. To our best knowledge, traditional transaction models, such as various advanced or long-lived transactions, lack a mechanism of choosing alternative participants. The whole transaction can only be successful after all participants succeed at a certain point. ABT does not require every participant to be successful. As long as there are available alternatives, the whole transaction goes on and in most situations this can save the expensive rollback or compensation of sub transaction modules. We believe it is more important to qualify the result, and provide a method to improve the quality of ongoing results over time.

Each sub-transaction is managed by a sub-agent. Each agent implements its own transaction model and recovery strategy. Resource locking is also completely controlled by a sub-agent and is effectively isolated from the coordinator. These factors suit the natural decentralized nature of the web services environment very well, and simplify the control logic of the coordinator.

4. CONCLUSION

In this paper we outlined ABT, an agent based web services transaction model. This model is a hybridization of web service, transaction and intelligent agent architectures. In terms of web services, it is an orchestration of web services; in agent terms, it is a Multi-Agent System (MAS); in transactional terms, it is a distributed service-oriented transaction system. By engaging autonomous agents to represent heterogeneous participants and by orchestrating web services, we have effectively eliminated the need for a centralized authority and isolated local resources from the coordinator. Moreover, our alternative recovery strategy greatly reduces the cost of handling failure. Finally, the discovery and rating capability as outlined in our system, turns traditional fixed transaction processing into a dynamically self-improving system, able to achieve higher and higher quality results as the system continues in operation.

Our ABT system is still at the prototype stage. We need to further develop and improve it as follows:

- Improve the discovery and rating algorithm, enabling it to handle more unexpected events.
- Include a programming framework to let our agents automatically wrap WSDL-described web services, rather than the current method requiring programmer intervention, to wrap SOAP RPC and WSDL-described agents.
- Make our internal transaction message protocol interoperable with open standards as they emerge.

The recent emergence of the web services world is allowing us and others to shed more light on the agent paradigm and to spread its adoption to the distributed transaction community. We argue that using agent technology to manage and orchestrate transactions is the right choice in the web services world, as opposed to all parties needing to agree at the outset, on what part their individual web service will play, in every particular orchestration it is involved in. If web services are to become the programmatic interface to the Internet, the automated equivalent of the browser interface in scope and popular adoption, then they need to be as decentralized and autonomous as possible. Applying autonomous intelligent agents to web service orchestration, maximizes the likelihood of that situation prevailing.

REFERENCES

1 Albornoz, J., Finding your way through Web service standards, Part 1: Will my Web service work with your client? 2002, IBM developerWorks.

2 Antonellis, V.D. and B. Zonta. Modelling Events a Data Base Application Design. proceeding of 7th International Very Large Data Bases Conference. 1981. Cannes, France: IEEE Press.
3 Astrahan, M.M., et al., An Overview of System R: A Relational Database System. IEEE Computer, 1979. **12**(4): p. 43-55.
4 Biliris, A., et al. ASSET: a system for supporting extended transactions. Proceedings of the 1994 ACM SIGMOD international conference on Management of data. 1994. Minneapolis, Minnesota, United States.
5 Buhler, P.A. and J. M.Vidal. Toward the Synthesis of Web Services and Agent Behaviors. Proceedings of the First International Workshop on Challenges in Open Agent Systems, AAMAS'02. 2002.
6 Burg, B. Agents in the World of Active Web-services. Second Kyoto Meeting on Digital Cities. 2001.
7 Cavedon, L., Rao, A., Sonenberg, E.A. and Tidhar, G. Teamwork via Team Plans in Intelligent Autonomous Agent Systems, in Proceedings of the International Conference on WorldWide Computing and its Applications, vol 1274 LNCS, Japan, 1997, pp 106-121.
8 Ceponkus, A., et al., Business Transaction Protocol. 2002, OASIS Committee Specification.
9 Champion, M., et al., Web Services Architecture. 2002, W3C Working Draft 14 November 2002.
10 Chang, J.M. and S.K. Chang, Database alerting techniques for office activities management. IEEE Transactions on Communications, 1982. COM-30(1): p. 74-81.
11 Dayal, U., M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. Proceedings of the 1990 ACM SIGMOD international conference on Management of data. 1990. Atlantic City, New Jersey, United States.
12 Elmagarmid, A., et al. A Multidatabase Transaction Model for InterBase. Proceedings of the Sixteenth International Conference on Very Large Databases. 1990. Brisbane, Australia.
13 Fernandez, M.F. and S.B. Zdonik. Transaction Groups: A Model for Controlling Cooperative Transactions. Persistent Object Systems, Proceedings of the Third International Workshop. 1989. Newcastle, New South Wales, Australia.
14 Finin, T., L. Y, and M. J. KQML as an agent communication language. Proceeding of the 3rd International Conference on Information and Knowledge Management (CIKM'94). 1994.
15 FIPA, ACL Message Structure Specification. 2001, Foundation for Intelligent Physical Agents.
16 Garcia-Molina, H. and K. Salem. Sagas. International Conference on Management of Data and Symposium on Principles of Database Systems. 1987. San Francisco, California, United States: ACM Press.
17 Freund,T and Storey, T. Transactions in the world of Web services. IBM developerWorks, http://www-106.ibm.com/developerworks/library/ws-wstx1/, August, 2002.
18 Goschnick, S. Shadowboard: an Agent Architecture for enacting a sophisticated Digital Self, in Department of Information Systems. 2001, Thesis, The University of Melbourne: Melbourne.
19 Goschnick, S. A Whole-Agent Architecture that draws Abstractions from Analytical Psychology. Chengqi Zhang, Von-Wun Soo (Eds.): Design and Applications of Intelligent Agents, Third Pacific Rim International Workshop on Multi-Agents, PRIMA 2000, Melbourne, Australia, August 28-29, 2000, Proceedings.

20 Goschnick, S. Enacting an Agent-based Digital Self in a 24X7 Web Services World. Proceeding of International Symposium on Methodologies for Intelligent Systems (ISMIS), 2003, Maebashi, Japan.
21 Goschnick, S. and T. Jin, OBY1 - Online Booking sYstem V1 System Requirements Specification. 2002, Department of Information Systems, The University of Melbourne: Melbourne.
22 Gray, J. and A. Reuter, Transaction Processing: Concepts and Techniques, ed. J. Gray. 1993, San Francisco: Morgan Kaufmann Publishers, Inc.
23 Gray, J.N. The Transaction Concept: Virtues and Limitations. Proceedings of the 7th International Conference on Very Large Data Bases. 1981.
24 Haerder, T., A. Reuter, and P.o.T.-O.D.R.A.C.S.-. (1983). Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys, 1983. **15.**(4): p. 287 - 317.
25 HP, HP Web Service Transaction 1.0 Tech Preview. 2002.
26 Jiang, J., et al. CovaTM: A Transaction Model for Cooperative Applications. Proceeding of ACM SAC. 2002. Madrid.
27 Jin, T., OBY1 - Online Booking sYstem V1 Functional Requirements Specification. 2002, Department of Information Systems, The University of Melbourne: Melbourne.
28 Klein, J. Transaction Internet Protocol Version 3.0. Proceedings Of The Forty-First Internet Engineering Task Force. 1998. Los Angeles, California, USA.
29 Klein, J. and A. Reuter. Migrating transactions. Future Trends in Distribute Computer Systems in the '90s. 1988. Kong Kong.
30 Langworthy, D. et al. Web Services Coordination (WS-Coordination), IBM developerWorks, http://www6.software.ibm.com/software/developer/library/ws-coordination.pdf. September, 2003.
31 Madnick, S. and M. Siegel, Seizing the Opportunity: Exploiting Web Aggregation. MISQ Executive, 2001. **1**(1): p. 35-46.
32 Moss, J.E.B., Nested Transactions: An Approach to Reliable Distributed Computing, in PhD thesis. 1981, Massachusetts Institute of Technology, Cambridge, MA.
33 Nwana, H.S., Software Agents: An Overview. The Knowledge Engineering Review, 1996. **11**(3): p. 205-224.
34 Pu, C. and G.E. Kaiser. Split-Transactions for Open-Ended Activities. Proc 14th Internattonal Conference on Very Large Databases. 1988. Los Angeles, CA.
35 Reuter, A. ConTracts: A means for extending control beyond transaction boundaries. Presentation at 3rd Workshop on High Performance Transaction Systems. 1989. Pacific Grove, CA.
36 Wachter, H. and A. Reuter, The ConTract model. Advanced Transaction Models for new applications, ed. A.K. Elmagarmid. 1991, San Francisco: Morgan-Kaufmann.
37 Weiss, G., Multiagent systems : a modern approach to distributed artificial intelligence, ed. G. Weiss. 1999: MIT Press, Cambridge, Mass.
38 Wooldridge, M. and N.R. Jennings, Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, 1995. **10**(2): p. 115-152.
39 X/Open, Distributed Transaction Processing: Reference Model Version 2. X/Open Guide, November 1993,. 1993: X/Open Ltd.
40 Yezdi Lashkari, M.M., Pattie Maes. Collaborative Interface Agents. Proceedings of the Twelfth National Conference on Artificial Intelligence. 1994.
41 Zisman, M.D., Use of Production System for Modelling Asynchronous, Concurrent Processes. Pattern Directed Inference Systems, ed. Waterman and Yayes-Roth. 1978: Academic Press.
42 World Wide Web Consortium, Web Services Choreography Requirements, Working Draft, http:/www.w3c.org/TR/2004/WD-ws-chor-reqs-20040311, 11 March, 2004.