An Agent-oriented Architectural Framework for Modeling, Enacting and Managing Web Services

N.C. Narendra
Software Engineering Services (SES)
Hewlett Packard India Software Operations (ISO) Ltd.
29, Cunningham Road
Bangalore – 560 052
INDIA
ncnaren@india.hp.com

Abstract

The ever-widening reach of the Internet has led to the proliferation of web services. Many of them interact with each other and undergo service composition in order to offer more complex services. In order to realize this vision, however, appropriate infrastructure needs to be developed first. The two most critical components of this infrastructure are an overall conceptual architecture that can be practically realized, and a simple (yet expressive) language for service description and composition. There has been a lot of work in recent years on both the above components. However, most of the work has not focused on the fact that web services could be highly dynamic and adaptive.

In this paper, we focus on the first component, i.e., the conceptual architecture. Based on our earlier work on Agent Societies [JAIS], which we have called Flexible Agent Society (FAS), we propose an architectural framework where web services are modeled as agents and their interactions are modeled as workflows and multi-agent conversations. Using our earlier work on adaptive workflow and agents [AdaptAgent], we also show how adaptivity in web service interactions can be modeled and managed.

Hence our research will show how the FAS, via agent and adaptive workflow technologies, provides a sound framework for the entire web service execution lifecycle, viz., web services discovering each other, forming collaborations, enacting common business processes in order to meet mutually beneficial business goals, and adapting these business processes mid-stream in response to changing business conditions. We will also show we leverage prior work in Web Services Management in order to provide a management infrastructure for web services in the FAS, albeit with some important differences.

1. Introduction

The ever-widening reach of the Internet has led to the proliferation of web services. Many of them interact with each other and undergo service composition in order to offer more complex services. In order to realize this vision, however, appropriate infrastructure needs to be developed first. The two most critical components of this infrastructure are an overall conceptual architecture that can be practically realized, and a simple (yet expressive) language for service description and composition. There has been a lot of work in recent years on both the above components. However, most of the work has not focused on the fact that web services could be highly dynamic and adaptive.

Most web services development, especially in industry, is based on modeling web services using well-known component models such as J2EE, COM+, etc. Although these models are convenient from a programming perspective, they suffer from quite a few shortcomings [HPL]:

- They can only provide a static definition of web services and the compositions thereof
- Due to interoperability issues between different component models, web services described as per different models will not be able to communicate with each other easily
- They do not easily possess facilities for web services to discover each other, especially in an open environment such as the Internet

Hence an approach that accurately models the semantics and operational logic of web services, is needed. Software Agent technology, with its roots in workflow technology and artificial intelligence, provides just such an approach.

In this paper, we model web services as agents executing workflows in order meet common business goals. By leveraging and extending our earlier work on Flexible Agent Society (FAS) [JAIS], we present an agent-oriented architectural framework that can model the entire web services lifecycle, viz., service discovery → collaborations → workflow execution → adaptation to meet changing business conditions.
Managing web services is crucial in order to ensure that predefined Quality of Service (QoS) goals are being met [Sahai1, Sahai2]. In this regard, we show how our FAS can manage web services. This is similar to the “traditional” web service management approaches, with some important differences:

- The FAS provides a centralized “meeting point” for web services to discover each other and form collaborations. Its repository also serves as a UDDI-like registry [UDDI] for storing information on existing web services.
- Due to the different flexibility levels provided in the FAS architecture (to be presented later in Section 2.3), web service management can be implemented at several levels of control, from the highly decentralized (as in [Sahai1]) to the highly centralized (as in [WSM]).

Our paper is structured as follows. In the next Section, we will provide a brief introduction to the concepts necessary for the rest of this paper. Our FAS concept is introduced in Section 3. In Section 4, we set the context by showing how agents are best suited to represent web service interactions. In Section 5, we describe how Web Service Management can be accomplished using the FAS, and elaborate on the differences with “traditional” web service management ideas [Sahai1, WSM]. The paper concludes in Section 6 with suggestions for future research in this area.

2. Workflow and Agents

2.1 Workflow

We borrow our workflow definitions from our earlier work on adaptive workflow [ACM]: a workflow is modeled as a directed graph whose nodes are the tasks with edges denoting the flow of control between the nodes. Nodes are of two types – work nodes and route nodes. In work nodes actual task execution takes place, while route nodes are decision nodes that route the workflow information to the appropriate work nodes based on evaluation of certain rules. Edges are of four kinds – forward edge, loop edge, soft-sync edge and strict-sync edge. Forward edges depict the normal workflow execution, which is in a forward direction. Loop edges are backward pointing edges that are used to depict the repeated execution of loops.

The rules in the route nodes are mainly of the Event-Condition-Action (ECA) type. That is, each rule specifies a particular action to be performed, upon receipt of an event, provided the condition is met.

The “sync” edges are used to support synchronizations of tasks from different parallel branches of a loop, and they are of two types:

- A “soft-sync” edge is used to signify a “delay dependency” between two nodes $n_1$ and $n_2$, i.e., $n_2$ can only be executed if $n_1$ is either completed or cannot be triggered anymore. This type of synchronization does not require the successful completion of $n_1$.
- A “strict-sync” edge between $n_1$ and $n_2$ requires that $n_1$ successfully complete before $n_2$ executes.

Clearly, the use of such edges must satisfy some conditions:

- Redundant control flow dependencies between nodes and loops should be avoided
- A sync edge may not connect a node from inside a loop body with a node not contained within that loop

We can also define states for each node, depending on where it is during the workflow execution. Hence we define the following states for nodes: NOT-ACTIVATED, ACTIVATED, DONE, FAILED, SUSPENDED. The definition of these states naturally leads to the conclusion that we can define the state transition diagram that each node has to obey. Indeed, this means that the state of every node in the workflow instance, is governed by the transition rules in the state transition diagram (please see [ACM] for details).

2.2 Adaptive Workflow

In [ACM], we have developed a 3-tier approach to adaptive workflow management, which is based on the graph-based workflow model described in Section 2.1. Essentially, we have recognized that a workflow needs to have a schema, i.e., a basic definition of its structure, before it can be instantiated and executed. Moreover, since workflow is typically executed in business organizations, it is essential that it meet certain business goals. Therefore we see that business goals (essentially arising out of planning) lead to workflow schema being defined, out of which a workflow instance is generated for execution. Hence we have identified that adaptivity is basically of three types:

- **Adaptivity at instance level**: here, only the workflow instances need to be modified, perhaps to make them more efficient, or to make them easier to execute. This is represented in our 3-tier architecture by an Instance layer.
2.3 Flexible Workflow

Traditionally, workflow management has been regarded as being centralized, monolithic, rigid and not easily amenable to adaptation. Moreover, the monolithic nature of workflow has always implied rigid centralized control, thereby forcing all users to follow the workflow without question. However, this does not meet the needs of most web-service based systems today, due to their inherent dynamism and decentralized nature. To this end, in our previous work, we extended ideas from [OpenWater] and developed an architecture for what we called “flexible workflow support” [ISFI], containing the following ideas:

- Any workflow can be classified to belong to one of the following three “control levels” – **loose, medium or tight**
  - Loose workflows can be defined, started and adapted by any user (not necessarily the Workflow Administrator) in collaboration with the other participants in the workflow. Such a workflow can adapt itself as much as possible, depending on the need and on the participants.
  - Medium workflows are also defined, started and adapted by any user, but need approval from the Workflow Administrator, in order to ensure that they adhere to a particular workflow schema.
  - Tight workflows are the traditional workflows, which are defined, started and adapted centrally by the Workflow Administrator.
- Henceforth, in this paper, we will refer to the person defining the workflow (who can be any user or the Workflow Administrator) as “workflow definer”.
- Extending this to distributed workflow means that there are two categories of workflows, which can be at different control levels – the overall workflow and its constituent sub-workflows (which could be executed at different workflow servers) – resulting in nine different combinations for the three control levels.
  - If the overall workflow process is at loose level, this means that it is defined among the workflow servers in a peer-to-peer fashion by the workflow definer.
  - If the overall workflow process is at medium level, this will need a Central Coordinator (also known as CC – similar to that of Workflow Administrator) that will need to approve the workflow before it is executed.
  - If the overall workflow process is at tight level, then the CC itself is the workflow definer.
    - Hence the CC in this case is also an adaptive workflow system just as the one described in Section 2.2, with the important difference that it supports and manages the overall distributed workflow process, with the constituent sub-workflows being managed by (probably geographically separated) individual workflow servers.
- Here, we see that adaptation can take place at many levels – at the overall workflow level, and at the levels of the individual sub-workflows and their sub-workflows (if any), and so on. However, sub-workflows are always defined as “black boxes” at the level of their parent workflows, i.e., as single tasks for prespecified entry and exit criteria. Hence it is the responsibility of the individual workflow definers to adapt their sub-workflows should there be a need to do so, at their respective Instance, Schema or Planning layers. In other words, workflow adaptivity here is defined in a "top-down" fashion starting at the CC, and is dependent on the overall business goals.

2.4 Agents

Currently, there is not much consensus on what an "agent" is, and many definitions abound. For our purposes, we will combine the following definitions:

- [OMG]: "An agent is a computer program that acts autonomously on behalf of a person or organization"
- [Griss]: "an autonomous software component that interacts with its environment and with other agents"
Hence we define an agent as "a software component that acts autonomously on behalf of a person or organization, and is also able to interact with its environment and with other agents".

There are many interesting linkages between workflow and agents, which we will be exploiting in this paper [Griss]:

- **Agents can collaborate to perform a workflow**, e.g., telecom provisioning, service provisioning, scheduling
- **Agents can be used to make workflow more intelligent**, e.g., by adding negotiation, reasoning or decision points
- **Workflow can be used to choreograph a set of agents**, e.g., application management
- **Workflow can be used to coordinate interaction between people and agents**, having agents delegate to people or other agents, e.g., telecom management system alerting a human operator, or assigning a repair or provisioning engineer

For our purposes, we recognize that the first two linkages represent *agent-enhanced workflow* (using agents to enhance workflow systems, i.e., agents representing workflow systems) and the last two represent *multi-agent conversations* (i.e., using workflow concepts to model agent interactions in multi-agent systems). Agent-enhanced workflow will be used to define what we will call "macro-workflow"; and multi-agent conversations will be used to model what will be termed "micro-workflow". Hence "macro-workflow" will model Adaptive Workflows, whereas "micro-workflow" will model Multi-Agent conversations.

### 2.5 Web Services

A web service is a service or application accessible via the Internet, which performs tasks, solves problems, or conducts business transactions [Sahai1]. The main features of a web service are:

- It is accessible via the Internet at a particular URL.
- It can be composite in nature (i.e., composed of several web services itself).
- It can be implemented via any one of several implementations such as CORBA, BizTalk, COM, etc.
- It can be dynamically created, modified or composed with other web services.

Hence web services hold out the promise of providing “plug-and-play” service composition, thereby promoting the use of outsourcing of specific functions to web services by organizations. Moreover, due to the "plug-and-play" facility, organizations can dynamically change any collaborating e-service in case, for example, the web service in question is not delivering as per the presupposed goals. For this reason, web services have been generally hailed as an important technology for B2B integration [Hagel].

### 3. Flexible Agent Society (FAS)

Our Flexible Agent Society (FAS), is based on the Contractual Agent Society (CAS) idea of [Dell]. It envisions a FAS as a collection of agents which come together to collaborate in order to meet common business goals. The flexibility arises from the idea that the agents are free to define and implement their own workflows at loose and medium level, in order to achieve common business goals.

The FAS provides the following features (the appropriate services provided by the Society Administrator is given in italics):

- There is a facility for admitting members into the society in an orderly fashion, i.e., there are rules and procedures for this (*registration, reputation as in [Dell]*) so that only the “right” members are admitted and in the “proper” manner
- Members of a society can “discover” each other in case they want to transact business with each other (*matchmaker as in [Dell]*)
- Once discovered, the Society Administrator facilitates negotiations among the members in order to reach an agreement about how to interact, and at what cost to each society member (*negotiation, loose agent-oriented workflow & multi-agent conversations*). Sometimes, the society itself may recommend/mandate certain procedures to be used (*medium and tight agent-oriented adaptive workflow*)
- Collaborations between the society members can also be defined in terms of a common goal that needs to be met, and also in terms of “contracts” [Dell], which are essentially commitments that each agent makes towards the Society (*Planning layer of Section 2.2*)
- The Society Administrator also monitors interactions between the members, measures their effectiveness based on certain predetermined parameters, and uses this information as historical data
for Simulation and Planning & Scheduling of future collaborations (Admin Module, Planning layer of Section 2.2)

Hence the conceptual architecture of our Flexible Agent Society (FAS) [JAIS] will be as in Figure 1 below (the green colored arrows depict the usage scenario described below):

Hence in our FAS, agents are free to collaborate with each other as per their own mutually defined workflows and multi-agent conversations as per the adaptive+flexible workflow approach described in Sections 2.2 & 2.3.

The registration (not needed for members already registered), matchmaker and reputation services as described in [Dell] are useful for our FAS, and we will incorporate them into the Interaction Layer. The matchmaker service will, upon request from a client agent, find the appropriate agents meeting the client agent's requirements. The reputation service basically monitors the extent to which the member agents have fulfilled their goals and contracts, and assigns "trustworthiness values" (based on Quality of Service parameters, security constraints, etc.) to the member agents; this service can also implement access control mechanisms, such as ensuring that participating agents meet certain necessary security constraints for the needed collaboration. In other words, this layer will be used by the society members (i.e., participating agents) to get admission into the society, discover each other, and form collaborations. As part of the collaboration process, the agents create - based on the goals - the following:

- Contracts, which are commitments made by the agents towards the Society and the collaboration
- The workflows - both macro and micro - that they will follow as part of the collaboration. This can be at either loose, medium or tight levels. The inputs for this will come from the Reputation service, via the Planning & Scheduling and Goals & Contracts modules.

Once this is done, the agents then start executing their workflows and conversations, with the help of the Agent-oriented Workflow Layer. The Schema and Instance modules there are functionally the same as the Schema and Instance Layers of our 3-tier architecture, hence adaptation can also be managed in the same way. The other modules, such as FAS Repository, Goals & Contracts, Planning & Scheduling, and
Monitoring & Simulation modules, are also derived from our 3-tier architecture. The FAS Administrator therefore plays the same role as that of the Central Coordinator described earlier in Section 2.3.

The results of the workflow executions are sent to the Monitoring & Simulation module, which uses it for monitoring the workflow execution and alerting the agents and the Society Administrator in case of problems. This data is stored in the Repository, and the Monitoring & Simulation module also uses this data for building simulation models of the workflows and conversations. These simulation models can be used by the Planning & Scheduling Module for future workflow planning and scheduling.

As a practical realization of the FAS conceptual architecture, we have developed an agent-oriented adaptive workflow architecture called AdaptAgent [AdaptAgent]. In this architecture, we have defined multi-agent conversations as embeddable in the workflow executions as single tasks with prespecified entry and exit criteria. Hence, as per our graph-based workflow model, the outcome of these conversations would determine the future course of workflow executions. In [AdaptAgent], a layered architecture, similar to the one in [ACM], has also been defined for adaptivity in multi-agent conversations. Moreover, conversations can also belong to loose/medium/tight level, depending on the identity of the conversation definer (analogous to that of workflow definer in the macro workflow domain). There are a few significant differences, however:

- Conversations typically involve more than one party interacting with each other. Hence their structure needs to be quite different from that of macro workflows.
- Unlike macro workflows, adaptivity in conversations can occur much more frequently and rapidly; it is, after all, easier for an agent to change the “course” of a conversation than is it for a (human or automated) worker to change the course of a workflow. Due to this, conversations are typically much more flexible and “lightweight” than traditional workflows, and hence conversation management is quite different from workflow management.
- While individual macro workflow tasks possess state (see Section 2.1), state information in conversations is present only at the overall conversation level. This is because state information makes sense only for entire conversations.
- Due to the lightweight nature of conversations, it should be easier to adapt conversations extensively in mid-stream without the need to change the Conversation Schema, since such a change would be disruptive to all the participating agents. To that end, although we retain the original 3-tier architecture embedded in the FAS, i.e., Planning-Schema-Instance for conversations, we split the Schema layer into two sub-layers:
  - The “upper” sub-layer, will continue to be the Schema layer, and will interface with the Planning layer
  - The “lower” sub-layer will be called the Conversation Mode Sub-Layer (CMSL). Since a conversation can be started and successfully completed by several agents acting together, the conversation schema should be an unordered set of speech act sequences along with some composition rules that dictate how the speech act sequences should be composed. Hence, the conversation mode is a subset of the speech act sequences in the schema; if the total number of legal speech act sequences in the Schema is $S$, then there are theoretically $2^S$ possible Modes for a Schema.

4. Web Services, Agents & FAS - Setting the Context

Web services possess features common to most distributed software systems:

- It possible to specify declaratively their interfaces, i.e., the inputs that they accept and the outputs that they would provide in return.
- They can be composed into composite web services; this composition can also be specified declaratively, using their interfaces.

However, web services display some unique characteristics:

- They are meant to be composed and re-composed in a highly dynamic manner, hence static compositions as for traditional software systems will not suffice
- Due to the openness of the Internet, different web services could be residing on different platforms and their interfaces built according to differing standards; moreover, they would communicate as per different protocols
- Also due to the openness of the Internet, web services (unlike conventional software systems) may need to be discovered on the Internet before web service composition can be done.
Due to the above reasons, it is clear that traditional component models are not sufficient for representing web services (see [HPL] for details). Rather, our web service representation model should extend the component models in order to provide the requisite openness, flexibility and dynamism. Hence we feel that the agent model is highly suitable.

The Entish [Entish] language, currently under development, aims to provide an agent-oriented language for web service description and composition. We are currently extending the Entish language; our extension, called AdaptEntish, will possess constructs to represent our adaptive+flexible workflow architecture, and will also contain constructs for representing multi-agent conversations, building on FIPA’s ACL language [FIPA1]. Indeed, FIPA has also announced the formation of a Services Work Plan [FIPA2] that aims to provide agent-oriented descriptions of web services. Hence the need for representing web services as agents is gaining ground.

The table below shows how web services lifecycle activities map naturally onto the activities of usage scenario in our FAS.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Web Services Lifecycle Activity</th>
<th>FAS Usage Scenario Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web service publication onto registries such as UDDI [UDDI] for the purpose of facilitating discovery</td>
<td>Members joining the FAS</td>
</tr>
<tr>
<td>2</td>
<td>Web services discovery</td>
<td>Members “discovering” each other</td>
</tr>
<tr>
<td>3</td>
<td>Contract establishment among web services, including QoS parameter setting</td>
<td>Establishing collaborations among the agents via contracts</td>
</tr>
<tr>
<td>4</td>
<td>Web service execution planning</td>
<td>Workflow planning via multi-agent negotiations</td>
</tr>
<tr>
<td>5</td>
<td>Web service execution, including management of the executing web services</td>
<td>Workflow execution and monitoring</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Workflow adaptation in response to changing conditions</td>
</tr>
</tbody>
</table>

One major difference is the inclusion of the workflow adaptation activity, which is not a typical component of most proposed web services architectures today. The other difference is that, for activities #3 and #4 in the FAS, the inclusion of the 3 flexibility levels will provide a richer level of collaboration establishment and workflow planning as opposed to planning in traditional web services architectures, as we shall see below in Section 5.

5. Web Service Management in FAS

5.1 Preliminaries

The most popular technique yet proposed for managing web services, envisages a centralized management system [WSM] that instruments web service executions, collects the instrumented data, processes and displays it on a console in suitable formats. Such a system has three major characteristics:

- **It is centralized**, i.e., there is a central administrative facility that implements all the management functions, perhaps with the help of management agents in the case of distributed systems
- **It focuses more on low-level transaction data at the network and web server level**, such as web access time, network latency, etc.
- **It usually assumes that web service execution does not change too frequently in mid-stream**

A totally decentralized approach towards web service management has been described in [Sahai2]. This approach relies on a technique called distributed correlation to synchronize management data flowing in as a result of instrumentation. At any given instant of time web services form a tree or directed acyclic graph (DAG) with web services at the nodes and web service interactions as the branches. As these DAGs are created either statically or dynamically and web service transactions are performed management data can be exchanged among these web services via a predefined protocol. The management data contains two types of data: the correlator and the management information. The correlation information is needed to track a web service transaction (especially since web service interactions are not always guaranteed to be
synchronous) and correlate the collected management data. Management information at every level is then collated and aggregated at the end of the transaction.

We can see that the distributed correlation technique solves the problems associated with the centralized technique; moreover, it also allows for high-level workflow execution data to be transmitted. However, it goes to the other extreme and does not provide even the minimal level of control needed for web service management, since there is no global picture until the web service transaction is completed. In addition, it also does not accommodate workflow adaptation.

Hence what is needed is a flexible system that can measure high-level data at the workflow execution level, and also accommodate workflow adaptation. By “flexible”, we mean that the system should be able to offer the continuum from totally centralized to totally decentralized management.

From the description of the FAS in Section 3, we see that it can serve as the flexible web service management system that we need, depending on the flexibility level (loose/medium/tight). In particular:

- Since the FAS models and enacts business workflows, it can collect high-level workflow data via the Monitoring & Simulation module, which also processes it and stores it in the Repository.
- For workflows at loose and medium level, the management function can be handled by the workflow definer. At this level, data collection and correlation can be implemented as in [Sahai2], in a totally decentralized fashion, with only the aggregate data (i.e., that of the overall workflow) being sent to the Monitoring & Simulation module for storage purposes.
- Data collection and correlation for tight level workflows can be implemented directly by the Monitoring and Simulation module itself, as per the “traditional” way, i.e., as in [WSM], including collection of data at the individual task level itself.

For each task in the workflow the following would be the management data that would need to be collected:

- Task ID
- Task Parameters
  - Workflow Instance ID (i.e., ID of the workflow instance that this task belongs to; in case this is a sub-workflow represented in the overall workflow as a single task, then this ID will contain the ID of the parent’s workflow instance)
  - Predecessor tasks, if any (these include tasks connected by forward and sync edges; these can also include tasks “ahead” of the task in question, in case of a loop edge; )
  - Successor tasks, if any (these include tasks connected by forward and sync edges; these can also include tasks “before” the task in question, in case of a loop edge)
  - Workflow Instance ID of predecessor tasks, in case it is different (for at least one predecessor task) from that of the current Workflow Instance ID (this is needed in order to accommodate workflow adaptation)
  - Workflow Schema ID of predecessor tasks, in case it is different (for at least one predecessor task) from that of the current Workflow Schema ID (this is needed in order to accommodate workflow adaptation)
  - Output data generated as per prespecified QoS parameters, if any; the parameters could vary, depending on whether this is a work node or route node

The data items presented above in *italics* are needed to incorporate on-the-fly workflow adaptation.

Since the predecessor tasks would belong to a different workflow instance, their information needs to be recorded, so that the right data aggregation can be done.

For the overall workflow the following would be the management data that would need to be collected:

- Workflow Instance ID
- Workflow Schema ID
- Workflow Instance Parameters
  - Workflow Definer ID
  - Flexibility Level
  - ID of Contract from which workflow goals are derived
  - Workflow Goals specified as QoS parameters (these can be represented using formal languages such as QML [QML]); for each QoS parameter:
    - Name
    - Expected value
Actual value (i.e., output data generated; this will be a consolidation of the output data generated by each workflow task)

For micro workflows (i.e., multi-agent conversations), the data is similar:

- Conversation related data
  - Conversation Definer ID
  - Flexibility Level
  - Conversation Instance ID
  - Conversation Mode ID
  - Conversation Schema ID
  - Conversation Instance Parameters
    - Conversation State
    - Related macro-workflow instance IDs (i.e., the macro workflows in which this conversation is modeled as a single task as described in Section 3)
    - Conversation Goals specified as QoS parameters (these can be represented using formal languages such as QML [QML]); for each QoS parameter:
      - Name
      - Expected value
      - Actual value

Since conversations are modeled as single tasks within macro-workflows, only conversation-level management data is sufficient to be sent to the M&S module (not at the individual speech act level), regardless of the flexibility level of the conversation itself. This data can then be construed as individual task-level data from the macro workflow perspective.

5.2 Incorporating Workflow Adaptation

We have seen in Section 5.1 that workflow adaptation is accommodated by storing the instance ID of the predecessor tasks. The detailed mechanism for adapting workflows has been described in [JAIS]. Briefly, it consists of aborting or rolling back tasks so that the adapted tasks can be introduced into the workflow. Hence workflow execution will come to a temporary halt during workflow adaptation, i.e., when tasks are being aborted or rolled back. During this period, the appropriate management data on aborted and rolled back tasks should be sent to the workflow definer and/or Monitoring & Simulation module. This data is then incorporated into the already collected data for storage and presentation; for example, the FAS administrator and/or the workflow definer will need to be shown how the previous (i.e., before adaptation) workflow performed against the pre-specified QoS parameters.

If the workflow is at loose or medium level, then the aggregated management data up to the point when the workflow stopped executing, will need to be sent to the M&S module. In the case of tight level workflows, however, since task level management data is being sent to the M&S module, this data aggregation will not be needed.

For multi-agent conversations, the situation is slightly different. In this case, the data (as detailed in Section 5.1) is collected separately before and after adaptation. Hence the "before" and "after" data are to be sent separately, but as part of a single task within the macro workflow, so that appropriate analysis and reporting can be performed.

6. Conclusions and Future Work

In this paper, we have focused on the important research area of modeling and managing web services. We have shown how web services can be naturally modeled using agents. We have also shown how our earlier work on Flexible Agent Society (FAS) [JAIS], can be extended into a web services management architecture, albeit with a few crucial differences.

There are several opportunities for future work:

- Detailed Architecture Description and Implementation - our architecture needs to be implemented and experimentally evaluated. In particular, we are in the process of developing the following: a detailed data model of our system, XML descriptions of the different management data that need to be collected, techniques for synthesizing, storing and presenting the processed data.
- Planning & Scheduling - we need to explore how planning & scheduling of the workflows executed by the web services, can be modeled and implemented in the FAS. That is, the functions of the Planning & Scheduling and Monitoring & Simulation modules need to be specified in detail and implemented.
• AdaptEntish & FIPA extensions - we need to complete our work on AdaptEntish, and integrate it into our architecture. We will also be contributing to the FIPA effort [FIPA2] in integrating the agents and web services areas.

7. Acknowledgments
The author wishes to thank his manager, Srinidhi Srinath, for supporting his work.

8. References