International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 5 (2013) pp. 642-651
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Web Service Based Model for Inter-agent Communication in Multi-Agent Systems: A Case Study

Sachini S. Weerawardhana[1] **and Gaya B. Jayatilleke**[2]

[1]Department of Computer Science and Engineering, University of Moratuwa,
Katubedda, Moratuwa, Sri Lanka
sachini.sw@uom.lk

[2]Department of Computer Science and Engineering, University of Moratuwa,
Katubedda, Moratuwa, Sri Lanka
buddhinath@gmail.com

Abstract: **Service oriented architecture (SOA) is a widely used model for enterprise application integration, mainly due to the presence of accepted standards and tools required to implement web services. While the service oriented architecture provides an elegant model for interoperability, it does not define a computational model for building the endpoints of complex distributed services it exposes in a typical enterprise application. On the other hand, software agent technology provides powerful models such as the Belief-Desire-Intention (BDI) model that allows developers to build distributed systems with enhanced reasoning capabilities. We propose a web services based communication model for interagent communication allowing heterogeneous agents in a multi agent system to freely collaborate. Although, several attempts have been made at integrating web service and agent paradigms, the focus on such work has been on implementing protocol translations from the agent environment to web services and vice versa. Our approach differs from existing methods in that, we specifically focus on goal and belief set sharing in BDI based agents. By providing a web services based interface, we allow agents as well as non-agent based services in a distributed system to yield the benefits of agent technology. In doing so, we prove that it is feasible for these two different technologies to complement each other in building scalable and highly adaptive distributed systems. Applicability of this proposition is demonstrated by designing and implementing a prototype multi-agent system for flood forecasting.**

Keywords: BDI, Multi Agent Systems, Web Services, Agent Oriented Software Engineering

I. Introduction

Modern day enterprise application integration relies heavily on carefully choreographed composition of unique services. Owing to the presence of standardized models, tools and technologies, Service Oriented Architecture (SOA) has become the de facto standard for implementing interoperable distributed systems. As highlighted in [1], with the growth of a business, the number of clients seeking a particular service a business unit offers rises. This increases demands on processing units such as servers providing a specific service, leading to models of the nature of federated servers with composite services where information from different sources need to be pooled together.

Another inherent problem is the need for constant system changes to cater to the emergent requirements. Software Agents provide a model for building distributed systems as autonomous entities that behave in reactive and proactive manner in achieving the designated system goals. Agents are able to collaborate, learn, reason, share their beliefs and goals and work as a community. The challenge lies in embedding these agent capabilities into web services so that more “value added" web services can be developed. A common feature in most of the prior attempts to enable sharing of goals between agents is that they are tightly coupled with the agent development platform. For example in[1], Bordini *et al.* have implemented a system where Jason[2] agents were enabled to communicate goals among one another using AgentSpeak(L). However, little emphasis has been put on finding a methodology to do the same using technologies that are more platform independent. The effort here is to address this particular aspect with the use of web services technology leading to communication across heterogeneous agent systems. Further an integration of agent technology and web services can help create, flexible, adaptive systems that can easily absorb evolving user requirements.

Applications of multi-agent systems are diverse, ranging from mobile applications, e-learning to knowledge based systems. In [3], the authors present a mobile agent based system, which provides the functionality of choosing a university, according to a user specified criteria. The authors have focused on minimizing the decision making tasks of the human user, so as to arrive at a logical choice of a university, based on fuzzy logic and heuristics quicker, easier and at a reduced cost. Similarly, in [4] Stoyanov *et al.* have experimented on service oriented, agent based middleware architecture, which could be used in providing mobile e-Learning services. The authors present a case study, where the proposed layered architecture is deployed in a university environment, using the JADE agent platform. Similarly, SOA too has proven its applicability in multiple domains. An attempt in utilizing SOA in disaster recovery and emergency response is described in [5]. Since it's origin, SOA has evolved from silos-like architectures to more collaborative, dynamic systems, in which services communicate and share data to gain competitive advantages particularly in business domains [6]. A common feature in both agent based and SOA applications is that they are distributed, launched on multiple platforms, and need to evolve with changing requirements. In our research, we investigate the possibility of both these technologies to complement each other, when used in a distributed, heterogenous environment.

We assess the extent to which prior attempts to integrate web services with agent systems have been successful, by drawing comparisons between each approach and discussing the deficiencies present. We compare the work of Dickinson and Wooldrige in Nuin[7], JADE Web Services Integration Gateway (WSIG) [8], WS2JADE [9] and AgentWeb Gateway[10]. A common property present in all these systems is that the architecture is designed to support protocol translations between agents and web services. For example, in WSIG, a key design pattern involves implementing one to one mapping from agent communication language (ACL) to communication protocol web services use (SOAP), to facilitate message transmission between an agent environment to the web service environment. This requires the implementation of complex codecs, thus limiting the extendibility and simplicity of the design. Our approach aims at finding an alternative solution, which enables agents to communicate their belief-sets and goals through a web service interface, without the need for any encoding and decoding.

In [11] we look at how agent platforms built using the Belief-Desire-Intention (BDI)[12] model can be integrated via web services. Here we provide a more detailed account of that work. We propose a template for representing a goal and a publish-subscribe model for sharing goals across agents. In section two we look at related work, specifically inter-agent communication implementations of widely used BDI agent platforms. We also look at work related to goal sharing in agent systems. In section three we introduce the proposed inter-agent communication model for goal sharing. In section four we describe a case study that implements the proposed model in a proof of concept multi-agent system with heterogeneous agents. We also look at how the proposed publish-subscribe model performs in scaling the system with agents. We conclude and lists some of the future work in section five.

II. Related Work

We analyze the inter-agent communication aspect under a web service based model, so as to realize the feasibility of implementing a publish/subscribe[13] oriented agent-communication platform. Existing BDI agent development platforms provide inter-agent communication using a variety of techniques. Here we look at the communication methods available in three widely used BDI platforms, namely Jason [2], JACK [14], and Jadex [15].

A. Inter-agent Communication in Jason

Jason is a Java based interpreter for AgentSpeak(L)[1], which makes use of speech-act based inter-agent communication. In AgentSpeak(L), a message received by one agent from another typically has the following general structure:

<sender, illocutionary_force, content>

Here, the illocutionary_force is also referred to as the performative and can be one of `.tell`, `.untell`, `.achieve`, `.unachieve`, `.askOne`, `.askAll`, `.tellHow`, `.untellHow`, and `.askHow`. When this performative is received, the agent's reasoning cycle in Jason will interpret the message and act accordingly [2]. In the Jason API a default method(internal action) `.send` is available to send messages to other agents. The syntax to this internal action is as:

```
.send(recipient, illocutionary_force,
        propositional_content)
```

SACI (Simple Agent Communication Infrastructure) [16] is a built-in component of the Jason platform that facilitates inter-agent communication between Jason agents distributed over a network. SACI uses Java RMI (Remote Method Invocation) to communicate between physically distributed Jason agents.

B. Inter-agent Communication in JACK

JACK is a commercially available agent development platform that extends Java to support Agent Oriented programming. JACK allows developers to defined goals (events) and plans providing a BDI style reasoning cycle within each agent. In local communication, agent-agent messages are sent directly with the directive `#sends` or `@send` statement within a reasoning method inside a plan, with the recipient agent's name as a parameter. Similar to Jason, JACK treats message sending and receipt as events. The JACK communications layer known as the DCI network needs to be used to allow inter-agent communication. The DCI network is layered in such a way that different underlying transport mechanisms can be accommodated, leaving the developer free of implementation details related to network communication. Extensions or changes to the architecture can be provided via new plug-ins. New communication infrastructures can be attached by overriding the appropriate run time methods.

C. Inter-agent Communication in Jadex

Jadex achieves FIPA-compliancy[17] in way of being based on the JADE Agent Framework. JADE provides the platform architecture and the core services and message transport mechanisms as required by the FIPA specifications. In Jadex framework, plans are represented as Java classes, which extends abstract classes providing useful generic functions like sending messages, dispatching sub-goals, reading and altering beliefs. Agents are defined using XML based agent definition file(ADF), which specifies the initial goals, plans and beliefs of the agent. Inter-agent communication in Jadex is achieved through asynchronous message event passing. Message events in Jadex have dedicated message types which constrain the allowed parameters and parameter types of the message event. Furthermore, Jadex supports all FIPA message types (equipped with FIPA parameters such as sender, receiver, performative, content)

D. Agent-Web Service Integration Attempts

Nuin

Nuin [18] is an open source Java framework for building BDI agents with special emphasis on building semantic web agents. An attempt to facilitate knowledge sharing among agents is made by encoding meta-knowledge and adding them to the agent's knowledge base using RDF/OWL. Specifics regarding the implementation of knowledge sharing is still an unmet requirement in the Nuin platform.

JADE Web Services Integration Gateway

JADE Web Services Integration Gateway (WSIG) [8] from Whitestein Technologies AG Switzerland, is a standalone intermediary gateway service that offers transparent, bi-directional transformations between FIPA compliant agent services and web services. They have used a non-BDI platform JADE to host agents. The web services employ the WSDL/SOAP/UDDI stack. The gateway intercepts calls from agents to web services and web service clients to agent services and does the relevant conversions from one language to the other. Codecs are used to do the translations: e.g. WSDL to ACL and vice versa.

WS2JADE: Web Service Integration with Jade Agents

WS2JADE [9] is a similar approach in achieving the same objectives as in JADE WSIG. WS2JADE uses a layered approach to connect JADE agent components with web services. There are two distinct layers in WS2JADE: interconnecting layer, containing entities that connect the web services and agents together, and also the management layer that manages the entities that are involved in connecting agents and web services. The architecture uses a special type of agents called Web Service Agents (WSAG), who are capable of communicating with and offering web services as their own services. The ontology generator is responsible for ontology generation and management. It translates data and its structure from Web service WSDL interfaces into meaningful information for Agents. To translate web service transport messages (commonly SOAP) into agent ACL messages the Agents communicate with, the SOAP envelope is first projected into Java languages and then into ACL.

AgentWeb Gateway

The key feature of the AgentWeb gateway [10] is approach is that it enables integration of software agents and web services without changing their existing specifications at the cost of time taken for translations, which is negligible as compared to a transaction. The middleware lies between the two platforms; agents and web services. It has components to do the necessary translations between the service registries (Search Query Converter), service description languages (Service Description Converter) and communication protocols (Communication Protocol Converter).

A noticeable similarity in all these implementations is the use of an intermediate middleware (codec) to map agent level protocols to web service level protocols, one-to-one. This rigidity prevents architectures using such approaches to adapt to different use cases. Our approach differs from these methods, in that we extend the agent's internal abilities to be able to generate the messages that can be used to communicate with other agents through a web service interface.

E. Goal Sharing of Agents

Representation of goals and the ability to reason about them is the feature that makes agent-oriented software engineering a powerful tool in building autonomous, intelligent systems. Goals have two aspects: a description of the state, and a set of plans for achieving the goal as a procedure. The former is necessary in order to reason about important properties of goals, and the latter is necessary to ensure that goals can be achieved efficiently in dynamic environments. Goals determine the course of actions an agent can take within its operating surroundings. Goal sharing enables the agents to work inter-dependently empowering them to complete actions which may seem impossible otherwise. In a goal sharing enabled environment, losing one agent in the community does not result in the overall system functionality coming to a halt. Remaining agents can communicate among themselves and act accordingly to minimize the impact of losing a certain set of capabilities. Moreover, agents will have the ability to locate methods to achieve goals (also known as plans) that can complete a certain activity more effectively than what is defined in its local plan set. Agents from different implementation specifications can work together in the same multi-agent system, thus allowing the system to benefit from advantages each specification provides.

Several approaches have been developed for goal-oriented multi agent systems. In [19] Michael Winikoff *et al.* has proposed a theoretical framework for goals which integrate both procedural and declarative aspects. Their aim is to allow agent development platforms to reduce the gap between agent theories and actual implementations of agent based systems. Following this, PRACTITIONIST [20], a framework that adopts a goal-oriented approach to develop BDI agents and stresses the separation between deliberation process and the means-ends reasoning was proposed. PRACTITIONIST agents can be programmed in terms of goals, which then will be related to either desires or intentions according to whether some specific conditions are satisfied or not. PRACTITIONIST agents also have the capability of detecting and rectifying conflicts among agents' activities and objectives.

However, the work that has been done thus far is highly dependant on the agent implementation platforms. For example, in [20] the implementation is based on Jason agent development platform. Popular agent development platforms such as JACK [14] and Jadex [15] already support goal-oriented agent development. Coo-BDI [21], extends the BDI architecture allowing agents to cooperate via plan exchange in cases which no applicable plans are available for managing an event. The proposed extension facilitates adaptability of agents by sharing resources among other agents. Yet, there is the need to come up with a mechanism do enable goal sharing using a platform independent method such as web services technology.

We introduce an inter-agent communication methodology using a message broker service [22] as the communication model. We extend Jason agents' functionality to enable sharing of goals between other Jason agents using the proposed model. Our implementation uses mainstream technologies built on top of Apache Axis2[23] and communicates with agents using XML messages. We further explore the ability of this proposed communication platform to incorporate the message transportation mechanism for JACK intelligent agents, to prove that the said message communication methodology is independent of agent implementation platform. Following sections describe the proposed method and a proof of concept implementation.

III. Open Inter-agent Communication

A. Publish-Subscribe Architecture

Publish-subscribe (pub/sub) messaging architecture [13] is a commonly used inter-process communication paradigm, which enables end processes to communicate with one another using topic based, content based or type based subscriptions. Topic based subscription was the earliest notion of grouping messages in pub/sub systems. Here events are classified based on a keyword. For instance if publisher P transmits a message under topic T, the message is broadcast to all recipients who are subscribed to topic T.

In designing a pub/sub messaging architecture the initial step is to determine the content of the messages and the publish subscribe methodology (type based, content based or topic based) that would interest the communication endpoints of the system. Next, based on the topology integration, message delivery can occur via a message broker, a bus interface or point-to-point integration. In message broker variation, which we use in our implementation, messages exchanged among publishers and subscribers transit via a broker, who maintains subscription lists. Message recipients subscribe to these subscription lists in order to receive messages from the broker.

Here, we draw a comparison between agent architectures and the pub/sub based systems, i.e. the event driven model. Similar to pub/sub systems, multi-agent systems too can respond to events such as message receipt, and external events received as perceptions. Thus we anticipate a possibility of capitalizing on this similarity, by formulating a communication link between the messaging system and the multi-agent system with the objective of achieving goal sharing.

B. Proposed System Architecture

Based on the improvements made to WS-Messenger in [24] we opted to use the pub/sub messaging system OGCE WS-Messenger [22] in implementing the proposed inter-agent communication infrastructure. WS-Messenger provides basic pub/sub functions as specified by WS-Eventing [25] and WS-Notification [26] specifications. It also includes message mediation facilitated by a message broker. The broker is responsible for message routing between subscribers and publishers, based on their subscription specifications. WS-Messenger consists of a subscriber, a NotificationConsumer Web service interface, a publisher, NotificationProducer Web service interface, Notification broker, an underlying messaging system, and XPath based message filtering. OGCE Messenger is a free and open source messaging library implemented on top of Apache Axis2 stack.

1) Message Structure Design

The primary task in establishing proper communication among agents is to identify the structure of the message that will be sent to and received by agents. The proposed multi agent system will demonstrate communication of beliefs and goals of agents. Since the purpose of each of these agent constructs are different, we designed two different XML message structures for goal communication and belief communication.

Belief Message

Beliefs of agents vary according to the system they try to model. Since the proposed case study models a flood forecasting system, we decided to encode the required measurements in a message structure as a simple XML message with the tags, Observation - representing the weather parameter measurement, e.g. rainfall and river water level, Location - stating where the reading was recorded from, e.g. Colombo, Keliniya, Unit - representing the measurement unit that quantifies the reading, e.g. millimeters, Value - representing the numeric figure of the measurement, and Time specifying the date and the time that a particular reading was made.

XML based messaging structure facilitated by OGCE message broker allows different belief representations be communicated with ease. The key is to identify what information the agents wish to communicate as beliefs and designing a message structure that properly encapsulates the information.

Goal Message

```
<agentinteraction>
    <goal>
        <type></type>
        <label></label>
        <parameters></parameters>
    </goal>

    <topics>
        <publishtopic></publishtopic>
        <responsetopic></responsetopic>
    </topics>
</agentinteraction>
```

Above is the XML message structure we designed to be used as the message transport unit, that would communicate information about a goal to another agent. By studying the agent concepts represented in both JACK and Jason, we have identified similar properties which we can use to describe a goal. JACK provides programming constructs to implement test, achieve, maintain and insist goal types, while Jason supports test and achieve goals. A `label` is the name used to identify a goal. We assume the recipient agent already possesses a plan to achieve the goal specified by the sender. The message receipt is the event that triggers a goal. `Parameter` tags are used to feed data required by the goal to execute the plan it is associated with. `PublishTopic` tag is used to create separate channels in a pub/sub messaging architecture. `ResponseTopic` tag is used when a certain message received at a recipient requires a reply back to the sender. That particular response can be sent using this field back to the source.

2) *Broker Integration*

For the implementation of the pub/sub based messaging system peered by agents we only make use of the eventing service. Goal and belief sharing occur under a publish subscribe scheme. Eventing service exposes broker related services open like publish, subscribe, renew, unsubscribe.

The extendibility of Jason makes it highly customizable according to different requirements of different multi-agent systems. New functionality can be added to the system by way of internal actions. New internal actions can be created by extending the class `DefaultInternalAction` and overriding the method `execute`. In this experiment, since we need to deviate from the existing inter-agent communication method `.send(recipient, illocutionary_force, propositional_content)` we use classes that are extended from the class `DefaultInteralAction` to send and receive messages via OGCE Message Broker. Furthermore, message broker integration with agents, message definitions, reading configuration definitions by agents are also done the same way.

JACK Agent Language provides templates for class level constructs. By extending these templates user can implement new features for JACK agents. Here, we extended the template `Agent` to define a new JACK agent type, which can communicate with the OGCE message broker. This template can be used to add new methods required to publish and/or subscribe into channels. In order to handle messages received from the broker, we further created a new plan for the JACK agent by extending the class template `Plan`.

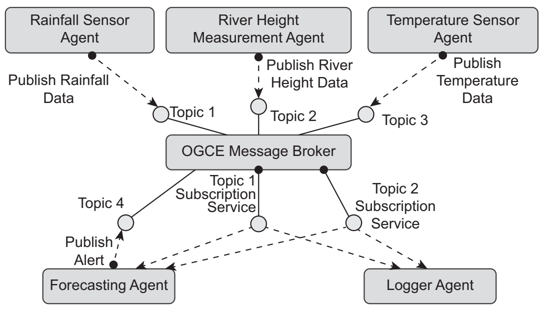

Figure. 1: Multi Agent System in a Pub/Sub Environment

Figure 1 illustrates the broker integrated system architecture. We exploit the concurrency feature present in both BDI agents as well as the OGCE messaging system. In that, we design the agents so that they can communicate with the message broker, as well as continue with their decision making activities at the same time. Further the Pub/sub messaging system is inherently asynchronous. Agents can acquire new functionality and/or knowledge without having to be concerned about how to communicate them to other agents. The message broker also can separately evolve in its design to handle multiple agents or agent systems, or in its internal system design.

IV. Case Study and Observations

A. The Background

As a proof of concept of the proposed goal and belief sharing model for BDI agents, we designed and implemented an agent based flood forecasting system. This agent based application models the flood forecasting process that is currently being used in Sri Lanka. We believe that this study would provide the expected motivation in the direction of improving the manual processes that are heavily being used in flood forecasting in Sri Lanka today.

The Hydrology Division of the Department of Irrigation in Sri Lanka is responsible for monitoring, collecting, processing, storing, dissemination of hydrological data and activating the flood warning system for major rivers. Kelani river, as at today is monitored by a flood forecasting system involving a lot of manual data reporting and recording procedures. Although, efforts have been made to automate this process, with the adoption of systems such as MIKE II [27], significant progress is yet to be reported.

Flood forecasting in Sri Lanka is done primarily based on two measurements; rainfall and runoff. Runoff measurements are taken only at rivers which have the tendency to overflow. As far as the country's commercial capital Colombo is concerned, flood threats occur mainly due to Kelani river, which is also one of the two most overflow prone rivers in Sri Lanka. Therefore to forecast flooding for Colombo area, they have established rainfall/runoff gauges in 11 areas in the Kelani river basin (figure 2). Additionally, they have reading stations established at all catchment areas upstream.

Floods occur at downstream river basin due to heavy rainfall in upper watershed, heavy rainfall in the locality, dam breach, or heavy rainfall in local plat areas (flash floods). Persons stationed at each location, report data to the central operation center in an hourly basis during a possible flooding period. If not, usually telephone calls are made out to all measurement stations daily at 0900hours 1200hours and 1500hours to obtain the readings. Forecast is made on the basis of present river water level of the location, river water level rising at up stream river gauge station and past 24 hour rainfall in upper watershed rain gauge stations.

Warnings are issued based on river water level. For Colombo district, the yardstick for issuing warnings is the reading obtained from the Kelani river gauge station at Nagalangam Street. According to the scale established by the Irrigation Department, a water level of 5 means that it is a *minor* flood level, where warning issuing begins. A water level of 7 represents a *major* flood level. A level of 9 represents a *dangerous* flood level and water level of 12 (reported only once in history) means a *critical* flood level. Warnings and alerts are usually issued to media, Disaster Management Center and local police. Readings from gauges are manually logged in record books. Each gauge station along rivers

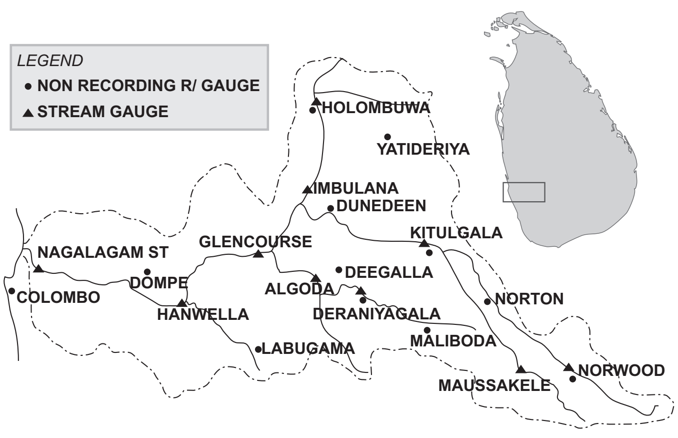

Figure. 2: Locations of Gauge Stations Alongside Kelani River

maintain their own logs and they are sent to Irrigation Department which acts as the log aggregating unit. Much of the present process is paper based and manual. As shown in figure 2, considering the locations of gauge stations, it is feasible to deploy a web service based system to automate forecasting process.

In the following sections, we discuss how this process can be automated using the proposed agent based system, supported by the service based communication infrastructure.

B. A Distributed MAS for Automated Flood Forecasting

With the proposed agent based system, equipped with the message broker infrastructure, we mimic a flood forecasting scenario. Trend of the readings obtained from data collection units are analyzed to predict if/when a safety threshold for a given gauge reading will be exceeded, and alerts are issued to the general public and media accordingly. The initial system design consists of five agents: Rain Gauge Monitor, River Gauge Monitor, Authorizer, Logger and Warning Issuer. The Rain Gauge Monitor agent and the River gauge monitor act as the sensors providing data into the system. The Authorizer agent collects these readings, analyzes and makes a prediction about the next possible river height. If this height coincides with one of the aforementioned alert levels, the respective alert is issued. The issuing of alerts is done via the Warning Issuer agent. The Authorizer agent, having made the decision to issue a warning communicates his goal (issue a warning) to the Warning Issue Agent, which in return executes the relevant plan.

Figure 3 illustrates the high level architecture of the multi agent system. We used the Prometheus methodology [28] to design the system. The external percepts of this system are rainfall, river height measurement and also addition of a new sensor agent for the system. The agents which are capable of perceiving the external environment are the Rain Gauge Monitor, River Gauge Monitor and The Authorizer agent respectively. This system comprises of 5 Jason agents and one JACK agent, which duplicates the behavior of the Warning Issuer Agent. We decided to incorporate JACK agents into this multi agent system to prove the platform independent feature of this proposed message communicating infrastructure. The messages that are exchanged between the agents are mediated through the message broker. When the requirement arises for the agent to communicate either a belief or a goal to another agent, the connection is made to the message broker's publish service under a commonly accepted URL (endpoint reference). We have allocated each monitoring agent a different topic to publish data under; e.g Rain Gauge Monitoring agent publishes data under the topic “Rainfall" while River Gauge Monitor agent publishes data under “waterlevelofriver" topic. The agents who wish to receive these data are Authorizer agent and the Data Logger agent. They both subscribe to these topics upon startup to receive data. In this prototype system, each monitoring agent records and reports a reading (rainfall and river water level measurement) every 10 seconds. We picked up this arbitrary interval value for simulation purposes although this choice deviates significantly from the real procedure.

C. Agent Decomposition

Rain Gauge Monitor Agent

Rain Gauge Monitor agent has one plan, according to which rainfall is measured and the reading is sent to the Authorizing agent and the Logger agent. As we mentioned earlier, the reading used here is a randomly generated number for demonstration purposes. In agent terminology the reading is a belief to the Rain Gauge Monitor. He then publishes this belief to the message broker under the topic “rainfall".

River Gauge Monitor Agent

The functionality of the River Gauge monitoring agent is similar to that of the Rainfall Monitoring agent. In that, it consists of a single plan, which generates and communicates a reading as a belief under the topic “riverwaterlevel" to the message broker.

Authorizer Agent

The Authorizer agent is responsible for analyzing the data in order to predict the next possible river water level depth. Additionally it creates a real-time graphical plot of the recorded rainfall and the river water levels, which is expected to assist human users. The Authorizer agent works in two modes: automatic mode allows the system to run without human intervention, the custom mode allows a user to retrieve past records of rainfall and river water level measurements.

Upon the receipt of the result message, the agent then runs an algorithm to calculate the next predicted value for the water level measurement of a river. The prediction is made by using time series analysis method, simple regression. Simple regression provides ordinary least squares regression with one independent variable estimating the linear model. We decided to use simple linear regression technique to make the predictions in water level measurements. Research on rainfall and runoff prediction models for Sri Lankan inundation areas are relatively scarce. However, in [29] and in [30] authors have conducted a survey and concluded that, based on historic data rainfall and run off can be sufficiently modeled using time series analysis with linear regression method. Therefore, based on these conclusions, we adapted a linear

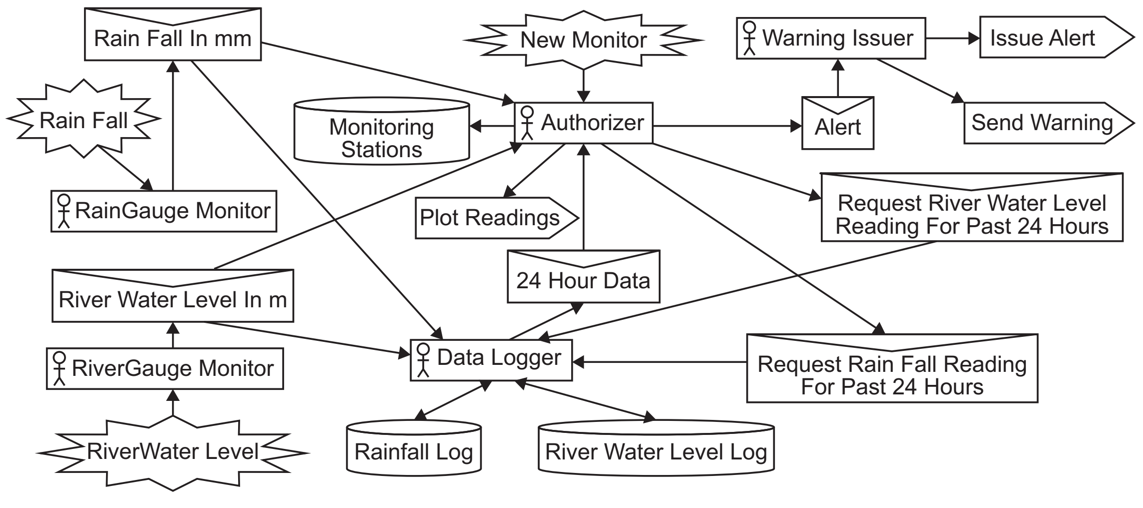

Figure. 3: System Overview Diagram of The Agent Based Flood Forecasting System

regression model to predict on the next expected river water level height measurement. The agent based design facilitates easy integration of different prediction models, to achieve different levels of accuracy with respect to the predictions. However, this is not the focus of our work.

Having computed the predicted river water level measurement, the Authorizer agent initiates another goal sharing phase. This is to inform the Warning Issuing agent that an alert should be fired. Plan “Notify Warning Issuer", which the Authorizer agent possesses takes care of this requirement. The Warning Issuer agent possess the plan to issue alerts. The Authorizer agent triggers this plan according to the prediction value it determines. In the manual process, the Irrigation Department processes the data, decides on when to issue alerts and sends alerts to police stations etc. In our agent based model this task is decomposed into two phases. Two separate agents: Warning issuer agent, Authorizer agent are responsible for achieving the goals sending alerts, and generating data required to issue alerts respectively. This design enhancement improves the agents’ chances of independently evolving in the future. In the sense, the Warning Issuer agent may incorporate new plans to send out notifications such as SMS alerts, email messages, manage an on screen interface to visually aid human users to locate areas under threats etc. The Authorizer agent may incorporate new prediction models for flood forecasting, add new metrics such as the wind speed, humidity, temperature to formulate more conclusive predictions and so forth.

The authorizer agent exhibits the goal sharing behavior at two distinct scenarios. Running under the custom mode of operation, human users can instruct the forecasting system to retrieve rainfall and runoff data of a selected time period. Upon receiving the required parameters, a goal sharing XML message is constructed. Then, the authorizer agent publishes this particular message under a topic that is known to the Data Logger agent. Plans to handle each query requests are defined in Logger agent definition file. The Logger agent subscribes to this goal sharing channel at startup so that it can respond to query requests as required. The second instance of goal sharing occurs with Warning Issuer Agent. Having formulated a prediction value for river water level measurement based on the past 24 hour data, the Authorizer agent makes a choice as to what alert plan will be executed under the current context. The actual issuing of the warning happens at the Warning issuer, hence plans to execute the warning is included at the Warning Issuer agent. The Authorizer agent, constructs the goal sharing XML message and publishes it on a channel to which the Warning issuer agent is subscribed. Upon receiving the message, the Warning Agent executes its plan accordingly.

Data Logger Agent

The Data Logger agent acts as the database of the system. The purpose of this agent to persist data recorded from the sensor agents. Presently, the Department of Irrigation maintains a lot of paper based logs, which makes maintenance extremely difficult. We propose data persistence with MySQL databases. This improves the system’s overall functionality by increasing accuracy, security, automated persistence, and ability to execute complex queries to retrieve data.

Warning Issuing Agent

The Warning Issuing agent is responsible for sending out warnings and alerts based on the decisions taken by the Authorizer agent. In order to examine the platform independent behavior of the communication infrastructure, we introduced two Warning Issuing agents implemented in JACK and Jason into the multi agent system. JACK implementation of the Warning Issuing Agent is composed of a single plan “Issue Warning" in order to handle a `BDIGoalEvent ReceiveAlert`. Jason implementation of the Warning Issuer agent is similar to that of the JACK implementation. AgentSpeak definition file consists of two plans; one subscribes the Warning Issuer agent to the service the Authorizer agent publishes warning data to, and the other plan enables the agent to reason and decide on type of flood alert (mild, critical, dangerous) that will be issued.

D. Performance Evaluation

In order to gain a quantified measurement on the multi agent system's scalability under the new web service based communication model, we looked at the performance impact when increasing the number of agents in the system.

We measured the average response time at the server as the number of sensor agents increased. As illustrated in figure 4, we observed a positive correlation between the average response time and the number of agents. As the agent count increases, the response time at the server also increases. It is interesting to note that the response times are in the range of seconds. This is mainly due to the message delivery delays in the network as well as the delays introduced at the application layer.

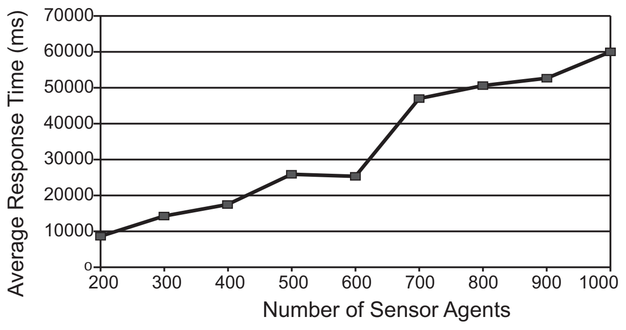

Figure. 4: Average Response Time vs. Number of Sensor Agents

Presently, the distributed agent communication system in Jason, SACI, has been discontinued owing to development issues present in the platform. Therefore we were unable to design an alternative distributed BDI agent communication platform to benchmark against our implementation. However, in the future we wish to extend the proposed BDI agent communication platform to incorporate other BDI agent platforms in the likes of Jadex, which has stronger support for inter agent communication. It will enable us to design and implement an alternative multi-agent system to benchmark our proposed implementation.

E. The Reliability Requirement

One of the problems in using a broker based service oriented communication model is the potential robustness issues that can arise due to broker failures [31]. We extended our experiment of the prototype system to observe the behavior of agents as the number of agents are increased.

As the number of agents in the multi agent system increases, we experienced packet loss at the server, (the broker). Packet loss causes the recipient agent to lose goal/belief sharing messages, thus causing plan failure. According to the graph we noticed that the packet loss is almost negligible up to 500 client agents. Beyond that range, the loss of data becomes more significant, edging up to 2% at 1000 client agents. Therefore, we recommend the multi-agent system running under our proposed communication methodology to have proper plan failure handling, especially considering scalability.

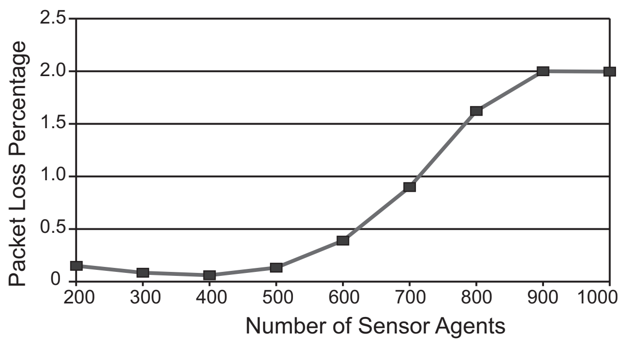

Figure. 5: Packet Loss Percentage Vs. Number of Sensor Agents

V. Conclusion

In this work we examined the feasibility of implementing an inter agent communication architecture based on web services to enable agents to share two agent level constructs, beliefs and goals. Using a service based approach also facilitated the interoperability of multi agent systems within real software environments like the Internet. We adopted a message broker service based model for the communication architecture. We extended Jason agents' functionality to enable sharing of goals between other Jason agents using the message broker service. By integrating JACK agents into a Jason based multi agent system, we also showed how a pub/sub communication model based on web services can be used to achieve collaboration amongst heterogeneous agents.

Use cases for practical applications of multi agent systems is a topic in discussion within the software agents research community. We intend to contribute to addressing this gap by developing a proof of concept system that practically implements the proposed communications architecture, which makes a strong case for using software agent technology in building smarter web services. The implemented multi agent system models the flood forecasting system in Sri Lanka and uses the proposed communication infrastructure to facilitate goal and belief sharing among agents.

Our test environment showed that the system has to deal with message loss as the number of agents scale. As future work, we will look into methods with which robustness and other relevant quality of service properties can be incorporated into this design. Furthermore, we intend to extend the support of the proposed communication model to Jadex, allowing Jason, JACK and Jadex agents to communicate goals and beliefs. Our message structure design does not support the communication of plans at this stage. Possible avenue for further research could look into the ways of designing a message structure which can be used to communicate plans and context information associated with a plan, enabling a higher degree of collaboration and team work within the multi agent system.

References

[1] A. S. Rao, AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language, *Proceeding of the 7th European Workshop on Modelling Autonomous Agents*

in a Multi-agent World Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996, pp. 42–55.

[2] R. H. Bordini, M. Wooldridge, and J. F. Hübner, 2007, *Programming Multi-Agent Systems in AgentSpeak using Jason* John Wiley and Sons.

[3] S. Sankaranarayanan and L. Cox, 2012, Intelligent Agent based University Search & Admission System Android Environment, *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 4, pp. 035-042

[4] S. Stoyanov, I. Ganchev, D. Mitev, V. Valkanov, M. O'Droma, 2011, Service-oriented and Agent-based Architecture Supporting Adaptable, Scenario-based and Context-aware Provision of Mobile e-Learning Services, *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 771-779

[5] J. Leppäniemi, 2012, Domain Specific Service Oriented Reference Architecture Case: Distributed Disasters and Emergency Knowledge Management, *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 4, pp. 043-054

[6] M. H. Danesh, B. Raahemi, S.M.A. Kamali, G. Richards, 2012, A Framework for Process and Performance Management in Service Oriented Virtual Organizations, *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 5, pp. 203-215

[7] I. Dickinson and M. Wooldrige, 2005, Agents Are Not (Just) Web Services: Considering BDI Agents and Web Services, in *Proceedings of The Workshop on Service-Oriented Computing and Agent-Based Engineering: SOCABE2005.*

[8] D. Greenwood and M. Calisti, 2004, "Engineering Web Service - Agent Integration," in *IEEE Conference of Systems, Man and Cybernetics*, vol. 2, pp. 1918-1925.

[9] T. X. Nguyen and R. Kowalczy, 2007, WS2JADE: Integrating Web Service with Jade Agents, *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, vol. 4504/2007.

[10] M. O. Shafiq, A. Ali, H. F. Ahmad, and H. Suguri, 2005, Agentweb Gateway - A Middleware for Dynamic Integration of Multi Agent System and Web Services Framework, *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, IEEE Computer Society, pp. 267–270.

[11] S. S. Weerawardhana and G. B. Jayatilleke, 2011, Web Service based Model for Inter-agent Communication in Multi-agent Systems: A Case Study, *Proceedings of the Eleventh International Conference on Hybrid Intelligent Systems (HIS)*, pp. 698–703.

[12] A. S. Rao and M. P. Georgeff, 1995, BDI-Agents: From Theory to Practice, *Proceedings of the First International Conference on Multi-agent Systems*, pp. 312-319

[13] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.M. Kermarrec, June 2003, The Many Faces of Publish/Subscribe, *ACM Computing Survey*, vol. 35, pp.114–131.

[14] N. Howden, R. Rönnquist, A. Hodgson and A Lucas, ,2001, Intelligent Agents - Summary of an Agent Infrastructure, in *5th International conference on autonomous agents*.

[15] L. Braubach, A. Pokahr, and W. Lamersdorf, 2005, Jadex: A BDI-Agent System Combining Middleware and Reasoning, in *Software Agent-Based Applications, Platforms and Development Kits*, Birkhäuser Basel, pp.143-168

[16] J. F. Hübner and J. S. Sichman, 2003, SACI - Simple Agent Communication Infrastructure, Available from: http://www.lti.pcs.usp.br/saci/

[17] Foundation for Intelligent Physical Agents, 2002, *FIPA Agent Communication Language Specifications*, Available from: http://www.fipa.org/repository/aclspecs.html.

[18] I. Dickinson and M. Wooldridge, 2003, Towards Practical Reasoning Agents for the Semantic Web, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, pp. 827-834

[19] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, 2002, Declarative Procedural Goals in Intelligent Agent Systems, *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pp. 470–481.

[20] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio, 2006, Goal-Oriented Development of BDI Agents: The PRACTIONIST Approach, *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology: (IAT '06)*, IEEE Computer Society, pp. 66–72.

[21] J. Omicini, L. Sterling, P. Torroni, D. Ancona, and V. Mascardi, 2004, Coo-BDI: Extending the BDI Model with Cooperativity,*Proceedings of the First International Workshop on Declarative Agent Languages and Technologies(DALT-03)*, Springer-Verlag, pp. 109-134.

[22] J. Alameda, M. Christie, G. Fox, J. Futrelle, D. Gannon, M. Hategan, G. Kandaswamy, G. von Laszewski, N. Gregor, M. Pierce, E. Roberts, C. Severance, M. Thomas, 2007, The Open Grid Computing Environments collaboration: portlets and services for science gateways, *Concurrency and Computation: Practice and Experience*, John Wiley and Sons, vol. 19, pp. 921-942.

[23] Apache Software Foundation, Apache Axis2/Java. Axis2 Home Page:http://ws.apache.org/axis2/

[24] R. Jayasinghe, D. Gamage, and S. Perera, 2010, Towards Improved Data Dissemination of Publish-Subscribe Systems, *Proceedings of the 2010 IEEE International Conference on Web Services, ser. (ICWS '10)*, IEEE Computer Society, pp. 520–525.

[25] World Wide Web Consortium (W3C), 2006, Web Services Eventing (WS-Eventing), *W3C Member Submission*, Available from: http://www.w3.org/Submission/WS-Eventing/

[26] S.Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, S. Tuecke, W. Vambenepe, and B. Weihl, 2006, *Web Services Notification (WS-Notification)*, IBM Corporation, Sonic Software Corporation, SAP AG, Hewlett-Packard Development Company, Akamai Technologies Inc, and Tibco Software Inc.

[27] K. Stronska, A. Borowicz, K. Kitowski, G. Michalik, G. Jorgensen, T. van Kalken, and M. Butts, June 1999, MIKE 11 as Flood Managemenet and Flood Forecasting Tool for the Odra River, Poland, *3rd DHI Software Conference*, Helsinger.

[28] L. Padgham and M. Winikoff, 2002, Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents, *OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pp. 97–108.

[29] H. Jayawardene, D. Sonnadara, and D. Jayewardene, 2005, Trends of Rainfall in Sri Lanka Over the Last Century, *Sri Lankan Journal of Physics*, vol. 6, Institute of Physics Sri Lanka, pp. 7–17.

[30] M. A. P. Desilva, March 2006, Time Series Model To Predict The Runoff Ratio of Catchments of The Kalu Ganga Basin, National Science Foundation, Sri Lanka.

[31] S. Kumar, P. R. Cohen, and H. J. Levesque, 2000, The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams, *In Proceedings of the Fourth International Conference on Multi-Agent Systems*, IEEE Computer Society, pp. 159–166.

Author Biographies

Sachini S. Weerawardhana received the Bachelor of Computer Science degree from the University of Colombo School of Computing, Sri Lanka in 2006 and the Master of Computer Science degree from the University of Moratuwa, Sri Lanka in 2010. She is presently a lecturer at the Department of Computer Science and Engineering, at the University of Moratuwa. Her research interests include, multi-agent systems, information systems, and Computer Science education

Gaya B. Jayatilleke earned his PhD in Computer Science from RMIT University, Melbourne, Australia in 2007. Prior to his PhD, he completed Master of Software Systems Engineering (MSoftSysEng) program at University of Melbourne, Melbourne, Australia in 2002/2003, receiving a High Distinction. He earned Bachelor of Science in Computer Science and Engineering degree from the Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka, in 2000, with a First Class Honors. At present, he is a post doctoral candidate at RMIT University, Australia. His research interests are Agent Oriented Software Engineering (AOSE), web based applications and web Services, Software Engineering process models/methodologies and Machine Learning.