Software Integration Using a Dynamic Wrapper Agent

MIRIAM A. M. CAPRETZ & MARY C. HRYB
Department of Electrical and Computer Engineering
University of Western Ontario
London, Ontario, CANADA N6A 5B9

Abstract: - Many application software packages still in use today are legacy systems which were developed decades ago in languages which are now considered obsolete. Yet, with proper maintenance and enhancements, these systems continue to perform their necessary functions effectively. These are not likely to be replaced any time soon, but present a restriction as they provide little interoperability in today's open environment. The option of replacing these systems to keep pace with ever changing needs is daunting, costly and may even be unnecessary. Instead, incorporating the services of these systems into an open, cooperative, distributed environment will not only extend their life and services but will also make them available to future systems which have not yet been created. Agent technology has been successfully used to provide legacy execution when embedded within the agent context. In order that legacy services become available to an extended clientele in the agent environment, a dynamic service providing connectivity to the legacy system is proposed in this paper. The focus of this solution is a wrapper agent that is able to create dynamic connections to various legacy applications on behalf of client agents as the need arises. It is proposed that the feasibility of the wrapper agent service depends on the agent's ability to internalize events and respond according to its goals and belief base.

Key-Words: - legacy systems, agent technology, back-box modernization, dynamic wrapper agent, interoperability

1 Introduction

The ever-increasing amount of legacy code still in use coupled with the constantly changing software environment has created a need for speedy methods of interoperability between existing systems and new technologies and systems. Let us consider that it has been estimated that we have approximately 50 billion lines of Cobol, roughly 80% of all software written since 1960, still in use in legacy systems. If we consider these legacy systems and what we currently must do to update their services to keep pace with an ever changing environment; we are left with the possibility of having an overwhelming number of either underperforming or possibly fragile software system resources. How can we ensure their integrity if we continue to update them with changes that will burden them beyond reasonable or reliable performance?
This concern identifies a need to provide the ability to integrate these useful, multiple, heterogeneous, existing software systems or sources with each other as well as with new software, services and technologies and doing this with minimal invasion to their integrity. One method of extending the life of these reliable, functioning, heterogeneous systems is to integrate them with new technologies which are able to use their resources while providing additional features demanded by our technologically oriented society. One such burgeoning technology is agent technology. Agent technology is also proposed here as a solution to the integration of the old with the new.

To provide continuous service in this rapidly changing technological society it is reasonable to explore the development of a dynamic wrapper which can provide a bridge between the old and new technologies as the new technologies evolve.

The dynamic wrapper agent proposed in the body of this work is based on the BDI (belief desire intentions) model of agency. The model is consistent with specifications put forth for agent platform interoperability. A prototype has been implemented in a JADEX agent using the JADE platform and written in Java and XML.

The remaining of this paper is organized as follows. The relatively new field of agents and agent systems is introduced and reviewed in section 2. This includes basic information on agent-based systems as well as the Belief Desire Intention model of agency. Section 3 presents the concept of the wrapper agent and reviews a subset of existing wrapper agent implementations. Application of specifications developed by the Foundation for Intelligent Physical Agents to the problem presented here is detailed in Section 4. Section 5 focuses on detailed information regarding the internal model of the dynamic wrapper agent developed, named DWrap, and its proposed implementation. Section 6 focuses on sample results obtained by using DWrap with a non-agent software system. Finally, conclusions are presented in Section 7.

2 Agent Technology

Software agents can be thought of as software components that operate on their own, or autonomously, not necessarily depending on human-user interaction. A widely accepted view presented by Wooldridge and Ciancarini [8] attributes the following basic traits to agent systems: autonomy, reactivity, pro-activeness, and social ability. Together these have been referred to as the weak notion of agency. A strong notion of agency includes the traits contained in the weak notion and the additional traits: mobility, veracity, benevolence and rationality.

Several agent technologies have been influenced by behavioural theories. Some are Agent Oriented Programming (AOP), Unified Theories of Cognition (UTC which lead to SOAR), Subsumption Theory and the Belief-Desire-Intention model [4].

2.1 BDI Model

BDI (Belief, Desire and Intention) is a mature architecture for intelligent agents. The BDI Model [4], based on the mental attitudes belief, desire and intention, was first introduced as a philosophical model for modeling rational (human) agents, but was later adopted and transformed into an execution model for software agents, based on the notion of beliefs, goals and plans. One way of modeling the behaviour of an intelligent agent is using the BDI architecture. Using the BDI approach, an agent's state is composed of *beliefs* (what the agent knows), *desires* (what the agent wants - also known as goals) and *intention (*how the agent intends to satisfy these desires - also known as plans).

Since BDI is a mature architecture which is incorporated into many agent models, it was decided to focus on this model of agency for this research.

3 Wrapper Agent – why an agent?

The wrapper agent can be considered to be a type of interface agent, which mediates between application agents of the new program functionalities and the existing legacy system. To the rest of the agent system, the legacy system can be wrapped to appear as an agent. This provides an ability to incorporate new functionality into the existing software.

The question could be posed, why not use a transducer? The difference is based on the idea of asynchronous behaviour. As an agent there is a choice to do one interaction before another. In synchronous behaviour there is a one to one mapping between input and action (output) and the output always goes to the requester. In agent behaviour, the agent does not necessarily send a response to the source of the message, and it may or may not take immediate action, depending on previous input. If it sends back results, it often does so asynchronously.

CIIMPLEX[5] and DIDE[2] illustrate some efforts on system integration. CIIMPLEX illustrates the usefulness of integrating semi-autonomous non-agent components into an agent system. However, this process of using agents to encapsulate functional modules requires knowledge of the internals of the non-agent program which is not always available. DIDE is specifically developed as a system application for engineering development integration. Hard connections are made between the systems to integrate them. This is not a dynamic integration system to which any new member could easily be introduced.

4 DWrap – A Dynamic Wrapper Agent

It can easily be appreciated that the dynamic integration service between agent and non-agent software systems must be further developed. An agent named DWrap (Dynamic Wrapper Agent) is presented here as a solution to this dynamic integration. The internal state and decision processes of DWrap are based on the BDI (belief, desire, intention) model of agency [4]. Using this model, the agent possesses a degree of flexibility in achieving its goals. It is able to decide internally how to reach its goals. Thus DWrap is able to behave in an asynchronous manner since its intentions are altered as its perception of the world alters.

DWrap serves as a channel for any number of other agents, which we shall refer to as the Client Agents. It supports multiple connections on behalf of possibly several Client Agents to different software systems simultaneously. Fundamentally, DWrap interfaces with a non-agent system on behalf of a Client Agent, which is unable to interface directly with the system itself. In this sense, it behaves mainly as a translator, but the functions of DWrap extend beyond translator to a manager of dynamic software integration services with multiple requests and responses.

DWrap would reside on an agent platform and is shown in the agent software integration reference model proposed by FIPA [3] in Fig. 1.

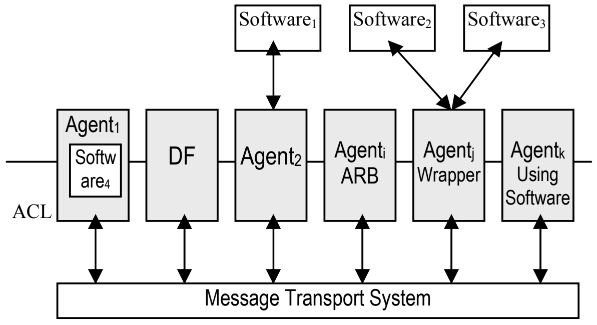


Fig. 1 FIPA Agent Software Integration Reference Model [4]

The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit association of companies and organizations sharing the effort to produce specifications of generic agent technologies to promote interoperability within and across agent-based applications [3]. In Fig. 1, based on FIPA specification [3], three methods of software integration, through Agent$_1$, Agent$_2$, and Agent$_i$ and Agent$_j$, are displayed.

The dynamic integration solution, however, requires an understanding of two agent capabilities: agent resource broker (Agent$_i$) and wrapper agent (Agent$_j$). The services of these agents are required in order to realize this integration solution. As the name implies, the Agent Resource Broker (ARB) is an agent that brokers resources. These resources are brokered as a set of software descriptions to other interested agents. Since an agent may require something beyond its capability to achieve its goal, using the Broker's resources it is able to seek assistance from another agent or software. The ARB possesses information on the capabilities of agents and software along with their software descriptions that uniquely identifies them. The ARB advertises its service via registration with the Directory Facilitator (DF). This research acknowledges the contribution of the brokering service to this integration technique, but the design and implementation of the Agent Resource Broker is beyond the scope of this paper.

The wrapper agent dynamically interfaces with a software system uniquely identified by a software description. Client Agents communicate with the wrapper agent using ACL (agent communication language) messages. The wrapper agent invokes the operations requested on the existing software system. Additionally, the wrapper agent may have multiple connections to other software systems and act on behalf of several clients upon these systems.

This is shown in Fig. 1 where the wrapper agent (Agent$_j$) supports multiple connections to software systems simultaneously. The wrapper agent also has the ability to dynamically manage additional software devices. This wrapper agent is realized in this research as the agent DWrap.

In order for DWrap to be able to provide a wrapper service dynamically, there are certain assumptions that must be made to use the model proposed by FIPA.

The ARB must have a software service description for the needed legacy system and for other non-agent systems. This description defines the nature of the legacy system and how to connect with it. This solution depends on the existing legacy system providing correct interface information to the wrapper agent so that the agent is able to make a connection.

The internal decision processes of DWrap are of primary interest in this paper. It is the internal agent design of DWrap and its mechanism for plan selection that make this agent able to function in a dynamic way. It is due to the interest in DWrap's internals that this agent is based on the BDI model of agency.

5 DWrap - Design Internals

In order that DWrap provide a service that is able to manage several Client Agents as well as legacy sources, DWrap must display an ability to coordinate its behaviours and plans with its updated knowledge in such a way that its goals are being met reasonably considering the circumstances in the environment.

DWrap is a rational agent with the capability of choosing a course of action based on mental attitudes. These attitudes are modeled on the concepts of belief, desire and intention. Beliefs capture the informational attitudes, desires capture the motivational attitudes and intentions capture the deliberative attitudes of this wrapper agent.

The agent's internal reaction and deliberation mechanism is based not only on incoming messages but is also affected by internal events and newly adopted goals. In this way DWrap exhibits autonomy since it has the ability to make decisions about its actions and is not limited to simply reacting to external stimuli alone. Instead, its response to external stimuli takes into consideration its internal state and knowledge base as well. The results of DWrap's deliberations determine the events that are sent off. These events, in turn, are dispatched to either the already-running-plans and/or to new-plans from the plan library. In this sense, DWrap is also a reactive agent. Reactivity refers to DWrap's perception of and response to the events that occur in its environment. The current beliefs of the agent affect the deliberation and choice of plans. Additionally, when plans run, they are able to affect the belief base. This in turn can result in further internal events, taking up new goals and continued execution of new plans. Thus, if while trying to achieve a goal the conditions in the environment change, DWrap is able to respond to these changes and possibly discontinue its current activity either temporarily or permanently.

In Fig. 2, the abstract architecture for DWrap is presented. This architecture is based on the concepts of BDI as implemented in JADEX [6], an extension to the JADE Agent Framework. The reaction/deliberation mechanism presented in this model does not vary for individual JADEX agents. Rather, the individualistic behaviours of specific JADEX agents are determined by their belief, goals and plans.

JADE is an open-source, agent-based software development project initiated at the Telecom Italia Group Company (TILAB) [1]. It provides a Java-based implementation of an agent platform in compliance with FIPA specifications for interoperable, intelligent, multi-agent peer-to-peer applications. On the agent platform each JADE agent is a peer since it has the ability to send as well as receive communications.

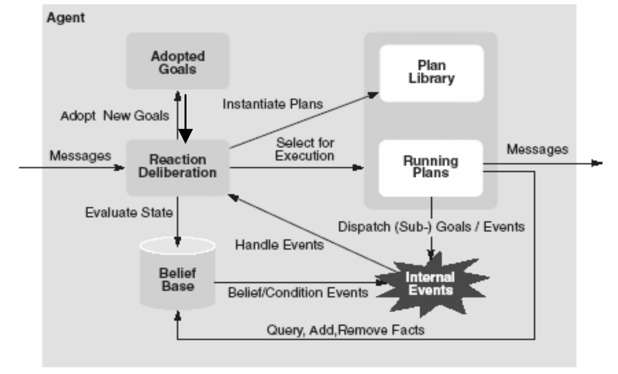


Fig. 2 JADEX Abstract Architecture [6]

DWrap is developed using the JADEX agent system that in turn was developed specifically to integrate with the JADE agent Framework.

The JADEX package provides the ability for the development of FIPA-compliant agents by applying the BDI architecture at the design and implementation layer.

DWrap has a belief base which is a store of information (or facts) which comprise DWrap's knowledge. Beliefs can be either single- or multi-valued. One function that DWrap is required to fulfill is that of translation from the client query into the legacy query. This translation information is stored as beliefs in DWrap's belief base. This belief base is implemented incorporating relational database concepts and an internal query language. A special feature of this belief base is its support for conditions. Beliefs can be retrieved and used for evaluation of a belief base state. Using a belief in support of a condition allows internal events to be generated when a condition is satisfied. This in turn may trigger plans or lead to new goals.

Goals are the states to be achieved by the agent or the motivation for the agent's behaviour. They are the driving force for the agent's actions. JADEX provides the ability for four types of goals: *achieve, maintain, query* and *perform*.

An *achieve* goal defines a desired state without specifying exactly how to get it. In this way the agent is able to exercise autonomy, trying various plans to achieve the desired state. This desired target state could be specified by an expression that is evaluated. A *maintain* goal requires monitoring of the state and executing those plans as necessary to re-establish the target state. It specifies that a state should be maintained once it is achieved. A *query* goal recognizes the need for more information. If there is a lack of information for the agent to make a decision, then plans are executed which assist in gathering the required information. A *perform* goal specifies a direct action and does not require the agent to perform any reasoning. It directly defines the plan to be executed.

A goal must first be adopted as an option for the agent to consider. Following this, there exist two alternative ways to implement reasoning in JADEX agents. Goals can be enabled (activated) or disabled (deactivated) based on internal conditions if a ruled based approach is used. Otherwise, activation and deactivation can also be achieved manually from procedurally implemented plans.

Plans are executed to achieve the goals. They are the procedures used to achieve the desired state and represent the actions that the agent can perform. There are two parts to the plan: the plan head and the plan body. The plan head is declared in the agents definition language while the plan body is realized in a Java class and can be threaded or non-threaded. DWrap's functionality is represented by separate plans (Java classes). These plans are in a plan library. Events that occur within DWrap trigger the appropriate plan(s). They are triggered in steps, where each plan step occurs after the event specific to it transpires. In Fig. 2 representing the internal architecture of DWrap, we see that already running plans have input from goals which then result in new adoptions to the plans. In this way, DWrap is able to continually accept new plans and cycle in the resulting newly adopted goals.

Plan selection and execution is guided by BDI-flags that capture the execution semantics of an active goal.

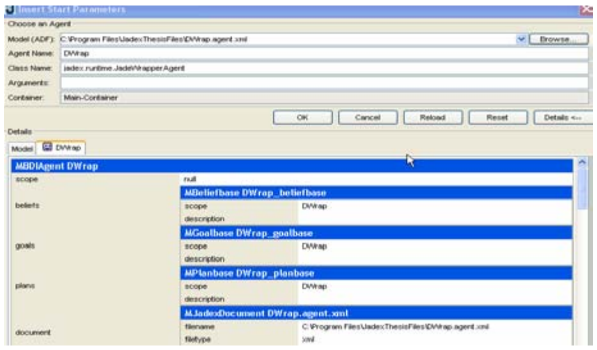
6 Implementation and Evaluation

This section presents an implementation of DWrap with an existing non-agent system called Air Gourmet [7]. Air Gourmet is a software system that coordinates airline food service management. Upon placing a reservation for a flight, the dietary requirements of each airline passenger are coordinated with the flight they are taking. The assumption under which this implementation is being made is that airline services are constantly being upgraded and Air Gourmet has undergone changes over several years that have resulted in it becoming less able to successfully accommodate additional maintenance changes. In particular, adding functionality to current implementation of Air Gourmet is necessary but costly.

In the system implemented, we are primarily concerned with the interactions between the Client Agent, DWrap and the legacy software system, which is Air Gourmet. A dummy agent is used to represent a client software system that requires contact with the legacy system. The represented client software system is an agent based system and sends a representative Client Agent to seek the services of DWrap in order that information can be exchanged with the non-agent legacy system, Air Gourmet. Thus, the dummy agent will provide the ability to send the required request to the Client Agent, which in turn interacts, with DWrap in order to interact with the legacy system.

DWrap is loaded onto the platform from the Agent definition file shown in Fig. 3. This file will provide the mechanism for plan selection by DWrap which is the core of its reasoning. Plans are selected not only for goals, but also for internal events and incoming messages.

A single Client Agent representing the client system is also loaded onto the platform to interact with DWrap. As seen in Fig. 4, the Client Agent and DWrap are now registered on the agent



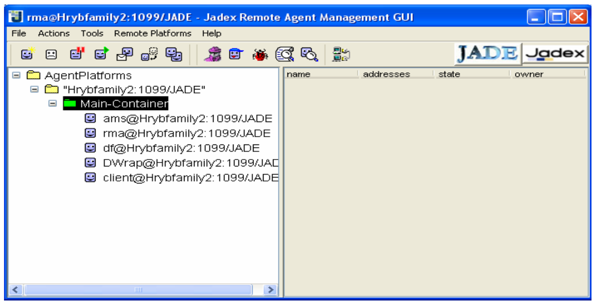
Choose an Agent
Model (ADF): C:\Program Files\Jadex\ThesisFiles\DWrap.agent.xml
Agent Name: DWrap
Class Name: jadex.runtime.JadeWrapperAgent
Arguments:
Container: Main-Container

Details

MBDIAgent DWrap		
scope	null	
beliefs	MBeliefbase DWrap_beliefbase	
	scope	DWrap
	description	
goals	MGoalbase DWrap_goalbase	
	scope	DWrap
	description	
plans	MPlanbase DWrap_planbase	
	scope	DWrap
	description	
document	MJadexDocument DWrap.agent.xml	
	filename	C:\Program Files\Jadex\ThesisFiles\DWrap.agent.xml
	filetype	xml

Fig. 3 JADEX Agent Creation of DWrap

platform and live there with the agent management system, remote monitoring agent and directory facilitator in the same agent container.


rma@Hrybfamily2:1099/JADE - Jadex Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help
JADE
Jadex

AgentPlatforms
"Hrybfamily2:1099/JADE"
Main-Container
ams@Hrybfamily2:1099/JADE
rma@Hrybfamily2:1099/JADE
df@Hrybfamily2:1099/JADE
DWrap@Hrybfamily2:1099/JAD
client@Hrybfamily2:1099/JADE

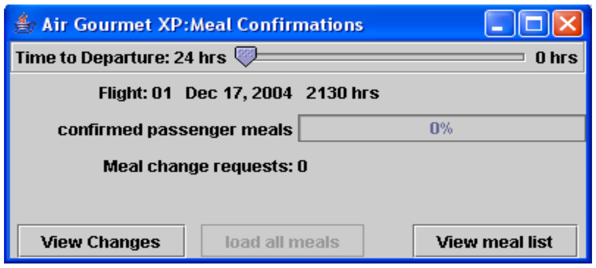
name	addresses	state	owner

Fig. 4 Agent Platform

As per FIPA, this implementation adheres to the specifications of software integration and expects that the Client Agent will have received a software description from the Agent Resource Broker that will provide a unique identification of the specific software system, Air Gourmet, which the Client Agent wishes to contact. In this implementation, the software identification provided to DWrap by the Client Agent is the physical location of Air Gourmet.

When DWrap is initialized, it begins to live on the platform and behave as a reactive agent. DWrap has as one goal to update its registration with the Directory Facilitator on a regular basis. This ensures that as its beliefs change and DWrap grows in knowledge, the services it is able to provide will be updated in the yellow page services provided by the Directory Facilitator. Thus, even before DWrap's services have been requested, it is driven by an internal event that is triggered by one of its registration goals.

Let us consider a scenario for the integration of Air Gourmet with a new agent based system called Air Gourmet XP. This system uses DWrap to extend the service of the original legacy system into an agent system that harnesses additional airline


Air Gourmet XP:Meal Confirmations

Time to Departure: 24 hrs 0 hrs

Flight: 01 Dec 17, 2004 2130 hrs

confirmed passenger meals 0%

Meal change requests: 0

Fig. 5 Air Gourmet XP Confirmation Screen

information and is able to display this in a timely fashion to airline personnel on a user-friendly interface (Fig. 5). This is extending the functionality of the legacy Air Gourmet by providing meal-updates to the caterer-provider of the airline meals and to the personnel responsible for loading and servicing meals. Air Gourmet XP provides, in addition to meal information, air flight information that was not previously integrated with the legacy Air Gourmet. As the time to departure approaches, the sliders bar approaches the 0 hrs mark. Additionally, if the flight is delayed, Air Gourmet XP captures this information and the bar is shifted away from the 0 hrs mark and back towards the 24 hrs mark accordingly.

Air Gourmet XP makes available the information originally offered by the legacy Air Gourmet system as well as new information in the form of updated passenger meal requirements, meal changes and flight schedule information.

DWrap has the ability to use the legacy Air Gourmet system and to extend its services in new functionality of Air Gourmet XP. The use of DWrap with the legacy Air Gourmet reduces the costs associated with replacement and eases maintainability of the system.

7 Conclusions

Due to the importance of existing systems, software maintenance has become the most costly stage of the software lifecycle. Thus, to reduce costs, there is a need in industry for software that is less expensive to maintain and with the ability to incorporate new functionality into it more easily.

The main objective of this work was to explore the integration of legacy systems with new agent technologies through the development of a dynamic wrapper agent. This agent would be required to meet the criteria put forth in FIPA specifications for dynamic integration of non-agent software systems with an agent environment. Furthermore, the wrapper was to internalize the BDI model of agency.

The decision to look at agent technology for a solution for legacy integration was mobilized by the view that agent technology is a key technology for supporting integration in heterogeneous open environments. The idea of a dynamic service was desirable to allow integration to be available to software systems and technologies not yet foreseeable. Like a file that can be shared amongst different software programs, the services of a legacy system would thus become available to many present and future systems.

Despite some limitations, the work presented in this paper is a step towards minimizing the costs associated with replacing what may be viewed by some as an obsolete or inflexible system. It will support the aim of decreasing maintenance effort. The possibility is that the human maintainer needs only to provide software interface information for the legacy system once and leave the deployment to DWrap, an implementation of a JADEX agent.

References:

[1] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE – A FIPA-Compliant Agent Framework", CSELT Internal Technical Report. Part of this report also published in Proc. of PAAM'99, London, April 1999, pp.97-108.

[2] DIDE Distributed Intelligent Design Environment, http://www.hds.utc.fr/~barthes/DAI-eng/projets/dide.html

[3] "Foundation for Intelligent Physical Agents (FIPA)", http://www.FIPA.org/

[4] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge, "The Belief-Desire-Intention Model of Agency", in Proc. of the 5th Int. Workshop on Intelligent Agents (V): Agent Theories, Architectures & Languages, pp. 1-10.

[5] Y. Peng, T. Finin, H. Chen, L. Wang, Y. Labrou, R. S. Cost, B. Chu, M. Russell, B. Tolone, A. Boughnnam, and J. McCobb "An Agent System for Application Initialization in an Integrated Manufacturing Environment", In Proc. of SCI'99/ISAS'99, Vol. 7, Orlando, FL., pp. 415-421.

[6] A. Pokahr, L. Braubach, "JADEX User Guide Release 0.92", Distributed Systems Group, University of Hamburg, Germany, 10, May, 2004.http://vsis-www.informatik.uni-hamburg.de/projects/jadex/features.php

[7] S. R. Schach, Object-Oriented and Classical Software Engineering, 5th Edition, McGraw-Hill, New York, NY, USA. 2002.

[8] M. Wooldridge, P. Ciancarini, "Agent-Oriented Software Engineering: The State of the Art," in Proc. of 1st Int. Workshop Agent Oriented Software Engineering, Sept 2000, pp. 1-28.