Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04), IEEE Computer Society Press

Agent Services-Based Infrastructure for Online Assessment of Trading Strategies

Longbing Cao, Jiaqi Wang, Li Lin, Chengqi Zhang
Faculty of Information Technology, University of Technology Sydney, Australia
{lbcao, jqwang, linli, chengqi }@it.uts.edu.au

Abstract

Traders and researchers in stock marketing often hold some private trading strategies. Evaluation and optimization of their strategies is a great benefit to them before they take any risk in realistic trading. We build an agent services-driven infrastructure: F-TRADE. It supports online plug in, iterative back-test, and recommendation of trading strategies. In this paper, we propose agent services-driven approach for building the above automated enterprise infrastructure. Description, directory and mediation of agent services are discussed. System structure of the agent services-based F-TRADE is also discussed. F-TRADE has been a online test platform for research and application of multi-agent technology, and data mining in stock markets.

1. Introduction

Stock traders can take less risk if they back-test their trading strategies in house first. Financial researchers often iteratively evaluate their found trading algorithms before publishing them. However, it is very hard for traders and researchers to build a system by themselves in order to evaluate their strategies. Another difficulty for them is that it is hard to get huge amount of real stock data for testing. All these issues can now be dealt with on our automated enterprise infrastructure F-TRADE (Financial Trading Rules Automated Development and Evaluation) [1].

F-TRADE is a web-based trading and mining support infrastructure. It supports online plug in of trading strategies or data mining algorithms and keeps privacy. Providers can iteratively evaluate these algorithms with online connectivity to huge real stock data from global markets, and even find some optimum strategy before going to markets. Public traders and investors can benefit from interested strategies on F-TRADE by subscription. Actually, F-TRADE also supports plug in of data sources, and system functional modules. Moreover, optimal strategies or optimal combination of input parameters for a specific trading strategy can be recommended on F-TRADE.

It is obvious that F-TRADE must be powerful enough for supporting the above functions. Furthermore, the following techniques should be provided in F-TRADE. It must be automated and flexible, so that it can support intelligent plug in of algorithms and system modules, optimization of parameters, interfaces generation, and presentation of outputs, etc. It must also support integration of distributed and heterogeneous data sources. Online extraction and transformation of data must be provided on colorful requests from users. Therefore, F-TRADE must be a practical support platform living in a real environment. Such a platform should be an automated enterprise infrastructure.

Agent-oriented methodology is very suitable for designing such kind of flexible and complex software system [2,3]. As a new research direction, agent service technology [4] is very promising for building enterprise automated systems. So, we investigate agent services-driven approach for building F-TRADE.

In this paper, we introduce the agent services-driven approach. We discuss how to set up an agent services-based automated enterprise infrastructure for trading and mining support in real and complex environment. First, modeling of F-TRADE is introduced in Section 2. A representation of agent services is proposed in Section 3. In Section 4 and 5, we describe directory and mediation of agent services, respectively. Section 6 presents the agent service-based architecture of F-TRADE. We conclude this study and discuss about future work in Section 7.

2. Modeling of F-TRADE

The main technical features must be dealt with on F-TRADE are as follows. (i) It makes the online plug-in of new trading strategies/mining algorithms, data sources, and system modules as flexibly and automatically as possible. (ii) It supports user personification-oriented requirements and interfaces of different system actors as custom-built and efficiently as required. (iii) Web-based service subscription, online connectivity with multiple markets, and user-defined data sources must be supported. (iv) It supports extension and re-construction of system as adaptively as possible. In summary, F-TRADE must provide (i) data service support, (ii) algorithm service support, and (iii) system service support.

Part of use case of F-TRADE is shown in Figure 1. We use Agent Unified Modeling Language (AUML) [5] technology to model the agent-based F-TRADE. Packages are used for describing the agent interaction protocols [6].

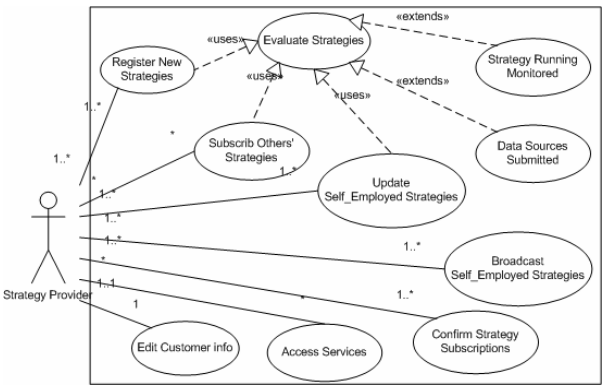

Figure 1. Partial use case of F-TRADE

3. Agent service description

Naming and representation of agent services are basic work in agent service-driven approach. We use multiple attribute items to describe an agent service.

Some basic attributes are as follows. *Locators* of sender and receiver indicate location of agent services, respectively. *Owner* is the one who holds the agent or service. Attributes related to service transportation are Type, Address and Message. *InputVariables* and *OutputVariables* are in/out parameters. *Preconditions* and *Postconditions* define constraints on executing a service. *SemanticRelation* defines semantic relationship between a set of *InputVariables* and a set of *OutputVariables*.

Key-value tuples are used to represent attributes in agent directory services. Furthermore, each item atom may have a list of items, which is called Item List. For instance, an item list of j inputs is a list of j (j = 1) items as <I>k_1:v_1, k_2:v_2, …, k_j:v_j</I>.

For each attribute in a service, there may be zero, one, or many items as required. Some of these items are mandatory, while others are optional. A token /O is marked at the end of all optional items for distinguishing mandatory items.

The following is example of an agent service, which registers a *Trading Strategy* algorithm agent.

<S> *RegisterAlgo: register(ParaArrayList)*
<SO>*ServiceOwner: PLUGINPERSON* </SO>
<ST>*ServiceType: Action*</ST>
<SL>*SenderLocator: AgentTransport.getSender()*</SL>
<RL>*ReceiverLocator: AgentTransport.getReceiver()*</RL>
<I>*InputVariables: AlgoInterface.getParameter()*</I>
<IC> *Preconditions: AlgoInterface.getInConstraint()*</IC>
<O>*OutputVariables : AlgoInterface.getResult()* </O>
<OC> *Postconditions: AlgoInterface.getOutConstraint()*</OC>
<IO>*SemanticRelation: one-one*</IO>
<TT>*TransType*: *AgentTransport.getTransportType()*</TT>
<TA>*TransAddress*: *getTransportAddress()*</TA>
<TM>*TransMessage*: *AgentTransport.getTransportMessage()*</TM>
<E>*Exeception*: *FTradeException*</E>
</S>

4. Agent service directory

With representation of agent services, the remaining work includes defining mechanisms for locating, directing, and transporting agent services in the agent space. Directory service is defined for managing directory. It can be described from domain ontology and follow generic specifications of FIPA namespace [4]. Agent service directory includes interfaces which must be implemented by an agent service.

Agent locator interface is as follows.

```
public interface AgentLocator{
   int hashCode();
   String getType();
   String getAddress();
   setType(String type);
   void setAddress(String address);
}
```

Agent service transport interface looks as follows.

```
public interface AgentTransport extends Transport{
    AgentLocator getSender();
    AgentLocator getReceiver();
    String getTransportType();
    Locator getTransportAddress();
    Message getTransportMessage();
    void setSender(AgentLocator sender);
    void setReceiver(AgentLocator receiver);
    void setTransportType(String ttype);
    void setTransportAddress(Locator taddress);
    void setTransportMessage(Message tmessage);
}
```

The following is part of the interface of agent service directory.

```
public interface AgentDirectory extends Directory{
    AgentDescription[] getAgentDescription();
    Vector getDirectoryEntry();
    void register(AgentDescription ad) throws
      DirectoryFailure;
    void update(AgentDescription ad) throws
      DirectoryFailure;
    void delete(AgentDescription ad) throws
      DirectoryFailure;
    void execute(AgentDescription ad) throws
      DirectoryFailure;
    AgentDescription[] search(AgentDescription ad)
      throws DirectoryFailure;
    void setDirectoryEntry(Vector de);
}
```

Algorithm interface is as follows.

```
public interface AlgoInterface {
    void setParameter(String parametername, String value)
      throws ftradeException;
    void execute() throws ftradeException;
      String getResult(String outputString) throws
      ftradeException;
    void setDataSourceDriver(String driver) throws
      ftradeException;
}
```

Address management and transport resolution of agents and services is organized by agent/service locators. Agent locators contain agent transport descriptions and transport type, and further transport specific address and properties of transport-type dependent, which can be used to locate an agent or an agent service.

The agent can access agent directory services to discover and locate target agent and services with which the source agents want to communicate. In the process of searching for the destination agent, the agent will technically retrieve the agent/service ontology library and directory services for an agent/service directory entry in the form of KVT. This agent/service directory entry encloses transport information which can be parsed to locate the target agents and services according to business rules defined in the agent service directories.

5. Agent service mediation

In our agent service-based trading strategy assessment system, the mediation [7] of agent services is supported by the following two-level matchmaking strategy (shown in Figure 2). All agents and services are named, organized and routed in a uniform service naming and addressing space.

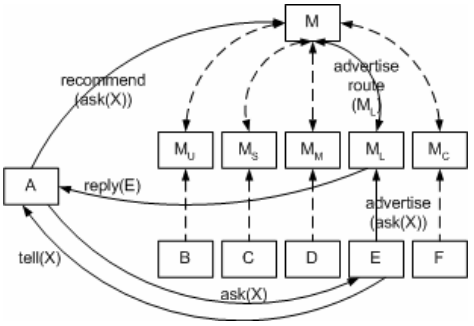

Figure 2. Agent service mediation

This mediation structure is generated adaptive to the organizational structure of the agent-based trading strategy assessment platform. In the F-TRADE, all user actors (for example, a trading strategy subscriber initiate an *ask(X)* request from the agent *A*) are interacted with the system console portal, in which there are five big functional centers: *User Center*, *Service Center*, *Administration Center*, *Algorithm Center*, and *Control Center*, each of them coordinates and administers most of issues related to the specific center. Correspondingly, five center-level mediator agents, M_U, M_S, M_M, M_L, M_C, are dispatched and lodging at the background of each of these five centers, respectively.

A center-level mediator agent, for instance, the M_L for the *Algorithm Center*, accepts and organizes advertising information including functions (for example, *ask(X)* performative) and location (parsing from the value of service attribute key, for instance, *ftrade.algorithm.tradingstrategy.vwap.vwapexecution strategy.execute*) of its belonging agents and services, and further reports the parsed location data to the high level root mediator *M*. The root mediator *M* organizes and maintains a global resource location list, and in the

above case is asked by agent *A* to recommend a mediator (for instance, M_L) which locates an appropriate agent who has advertised the service requested by *A*. Once the mediator M_L learns that agent *E* is willing to accept *ask(X)* performative, it replies to *A* with the name and location information of *E*. *A* is then free to initiate a dialog with *E* to request and response queries.

6. System architecture of F-TRADE

In order to make F-TRADE as an automated and enterprise infrastructure to support open, distributed and web-based operations and interoperability, the following concrete aspects have been investigated. The Gaia methodology [8] is used for multi-agent system analysis and design. Agent organizational framework and design patterns [9] are investigated in designing the global system structure and internal model of agents in F-TRADE. Java agent services [10] is the main technology for implementing agent services-driven F-TRADE.

Direct application components of financial trading and mining can be agentized in F-TRADE. These components must and can easily follow specifications of organizational framework and design patterns, and implement particular API for corresponding functional agents. The agents are plugged into F-TRADE through Control Center, and specified by naming, activity, location and directory in the organization, and relationship to other agents. Interfaces for user agent interaction are generated according to user profile and agent ontology.

One screenshot of F-TRADE is shown in Figure 3 There are five function Centers in F-TRADE. They are Administration Center, Algorithms Center, Control Center, Services Center, and User Center.

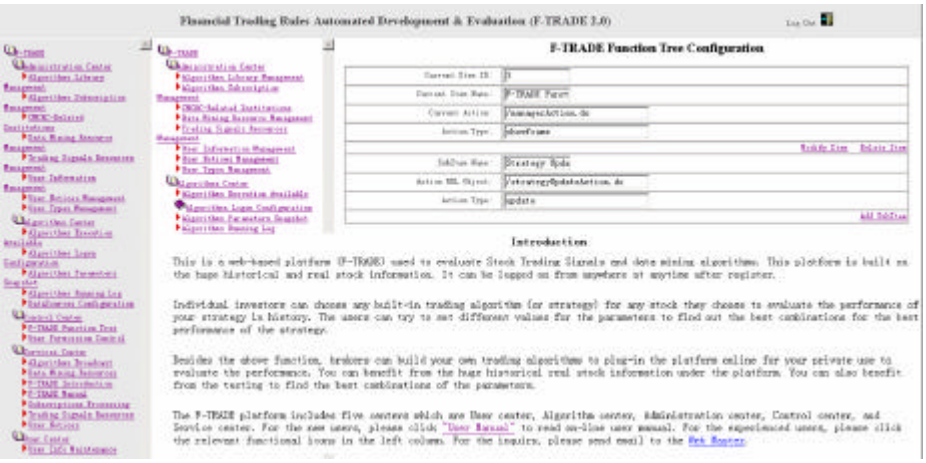

Figure 3. Screenshot of F-TRADE

Six types of users including algorithm providers and subscribers can interact with corresponding user interface agents lodging in the above Centers. User requests will be mediated and directed to target functional agents by mediator agents. The target agents will act on the requests with cooperation from other functional agents and services as required, or deliver requests to other relevant agents for further activities.

Currently data on AC3 [11] is connected online.

7. Conclusions and future work

In this paper, we have proposed an agent services-driven approach for building an automated enterprise infrastructure—F-TRADE. It is used for trading and mining support. We described description, directory, and mediation of agent services in F-TRADE.

F-TRADE has been running online for more than one year. With the agent services-driven approach, F-TRADE has evolved into research and application test bed in both the E-Intelligence Group of University of Technology Sydney [12], and the Data Mining Program at Capital Markets CRC [13].

References

[1] http://datamining.it.uts.edu.au:8080/tsap

[2] M Wooldridge, *An introduction to multiagent system*, Wiley, 2001.

[3] N. R. Jennings (2001) "An agent-based approach for building complex software systems" Comms. of the ACM, 44 (4) 35-41.

[4] FIPA abstract architecture specification, www.fipa.org.

[5] AUML website: www.auml.org.
[6] L.B. Cao, "Agent Services-Driven Plug-in Support in F-TRADE", submitted to 17th Australian AI, 2004.
[7] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT press, 2001.
[8] F. Zambonelli, N. R. Jennings and M. Wooldridge (2003) "Developing multiagent systems: the Gaia Methodology" ACM Trans on Software Engineering and Methodology 12 (3) 317-370.
[9] L.B. Cao. *Studies on some problems in multi-agents-based open giant intelligent systems*. PhD thesis, Chinese Academy of Sciences, 2002.
[10] Java Community Process. Java agent services specification. 5 Mar 2002.
[11] http://www.ac3.com
[12] http://datamining.it.uts.edu.au
[13] http://www.cmcrc.com