The DigitalFriend: *the first End-User oriented Multi-Agent System*

Steve Goschnick
University of Melbourne, Agent Lab
DIS, 5.39, 111 Barry St, Australia, 3010
61+3 0407 544 260
stevenbg@unimelb.edu.au

ABSTRACT

This paper details the *DigitalFriend*, a new form of Personal Assistant Agent for building complex personal systems, often incorporating simple services, including web services, but also including sophisticated intelligent software agents. The blueprint for the DigitalFriend is the ShadowBoard agent architecture, itself inspired by the *Theory of Sub-Selves* from Analytical Psychology. It is a 24-hour, 7-day (24x7) system, and uses a *role-oriented message lens* to filter and highlight the information emerging from the various sub-agents making up a user's *Digital Friend*, prioritized for their attention. Various agent types are available including one that wraps SOAP and WSDL web services, and another that retrieves RSS feeds. In addition to providing an umbrella technology over an array of individual services and functionality, the DigitalFriend can logically orchestrate numerous agents, with a built-in constraint logic language (CoLoG) interpreter, forming new synergies of functionality (i.e. mash-ups), often unforeseen by the individual service providers - all configurable from within a GUI interface, aimed at end-users.

Keywords

Interactive environment, Integrated development environment, Personal Assistant Agent (PAA), Agent-oriented interface, Web Service orchestration, Mash-up development environment, Visual logic language, RSS, SOAP, WSDL, Multi-Agent System, Intelligent agents, ShadowBoard agent architecture.

1. Introduction

The DigitalFriend is an implementation of the ShadowBoard agent architecture [3], designed to be a sophisticated but user-friendly multi-agent system (MAS). It is used to implement numerous sub-agents, which together forms a single, sophisticated complex autonomous agent in the more formal autonomous agent sense [1]. A distinguishing feature of ShadowBoard from most multi-agent system architectures, is that the non-autonomous sub-agent components are a first-order entity - from the theory up. This means that simple processes like SOAP [12] and WSDL [15] web services, and RSS feeds [10] are wrapped by relatively simple sub-agents, without concern nor a sense of compromise, that they are not fully-fledged intelligent agents. Conversely, sub-agents may well be as sophisticated as an expert system, an extensive Java program, a Relational DBMS application, or a BDI agent [11].

The *DigitalFriend* is a MAS that is used to build and operate a powerful *Personal Assistant Agent* for a given individual - their *digital friend* - to help them cope with the 24/7 information and cognitive workload – depending on the user's current activities. A design goal of the DigitalFriend is to automate the user's routine tasks and to provide expertise where they have weaknesses or lack focus (but are obliged to service commitments), allowing the user to concentrate on the things they like most and do best. It is also designed with *universal access* concepts in mind - technology that can be used by *anyone, anywhere, anytime* [5].

Figure 1 shows the opening screen of the DigitalFriend. The left panel of the split window, represents a hierarchy of agents and sub-agents, all of which are focused on doing some work or activity for the user, easing their load, making them more efficient in dealing with commitments, more focused on their areas of interest. The DigitalFriend is most effective where users need and

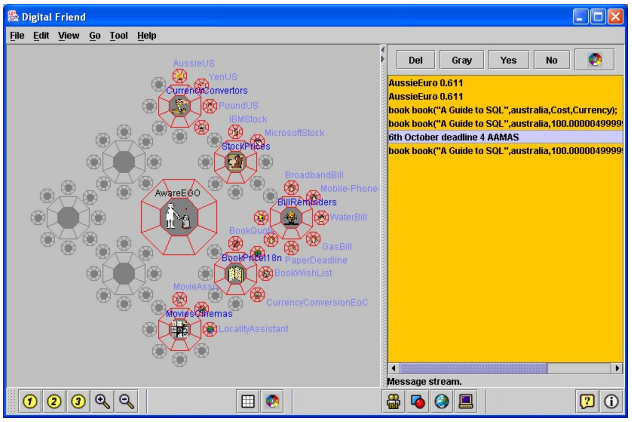


Figure 1. The DigitalFriend opening screen

want some semi-automated help. The multiple levels of the octagonal-tile-patterned interface, currently unique to the DigitalFriend, are called the FUN (*Friendly User Navigation*) interface. It is designed to make navigation of deep hierarchies of knowledge and process, easier and more intuitive to use, as demonstrated by Lane et al [9] who performed a *usability evaluation* of the FUN interface.

The right panel in figure 1 is the *message stream*, a chronological list of all the communications from the user's *society* of accrued agents, *monitoring, notifying* and *alerting* from a myriad of services, computational agents, DBMS and smart timers. There is no practical limit to the *depth* of the hierarchy, however, the width is limited to a maximum 8 children per generation, which is a design principle of FUN [9].

2. The *Bubble-Pond-Kelp* Metaphor

The message stream window can be thought of as the *surface of a pond*, the messages as *bubbles* on the pond surface, and the agents sending the messages, as the forest of *kelp*. Continuing with the metaphor, we do not want the pond to be boiling with bubbles, therefore each agent, in an individual's digital friend, can be told when it should send messages and what sort of messages the user would like brought to their personal attention. There are methods of filtering these messages, covered further down, including the use of a *role-based message lens*.

In the example in figure 1, the messages which have surfaced are: an up-to-date currency rate between the *Australian dollar* and the

Euro; the current cost of a book titled '*A Guide to SQL*' (from the user's wish list of books in a local datastore), priced at *Noble and Barnes* via a SOAP web service [12], displayed in Australian dollars (after a sub-agent has done a conversion from the original quote in US dollars); and a reminder of the deadline for papers to the AAMAS agent conference.

To allow the user to focus on the messages of most interest at any particular time, we use the role/sub-role categorization of sub-agents, as a *message lens* (see Section 2.2). This role hierarchy is taken from the underlying ShadowBoard agent architecture and methodology [3,5].

2.1 Inter-agent Chatter

The messages the user gets to see on the surface is just a fraction of the total messages being passed around between agents.

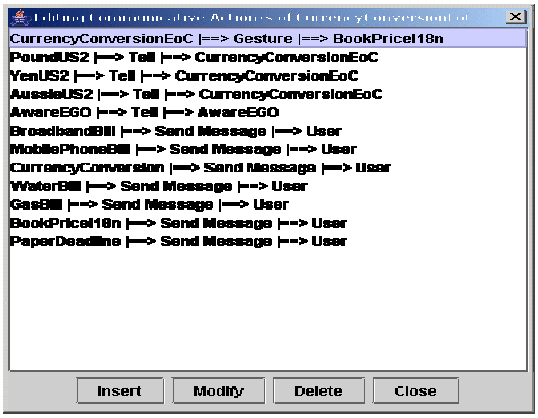


Figure 2. Inter-agent communication directives

Figure 2 displays a list of the *message paths* and message *types* between various agents and the user. I.e. The paths are constructed via the list of three-part communicative acts as follows: *sender* → *message-type* → *receiver*. Note: The user is considered to be just another agent, from the agents' point of view – often termed a *human-in-the-loop* by agent researchers.

In this list, the last line instructs an agent called *PaperDeadline* to send a *message* directly to the *User*. While the top line (currently highlighted) instructs an agent called *CurrencyConversionEoC* to send a *gesture* to a second agent called *BookPriceI18n*. This inter-agent communication is represented by an agent communication language (ACL), covered in more detail further down.

2.2 The Role-based Message Lens (RBML)

If a user's digital friend is made up of hundreds or even thousands of sub-agents, the number of messages that may *bubble up* to the message stream panel (the *pond surface*), could become overwhelming. So the DigitalFriend provides a message filtering system, using what is termed here a *role-based message lens,* to manage the *pond surface*. See figure 3, in which the *Initiator* role is being used to highlight messages from sub-agents associated with that role.

The RBML is based on the psychological notion that when we are doing *tasks*, we are generally working, acting or playing in some *role* or *sub-role* in our complex modern lives. These roles can be

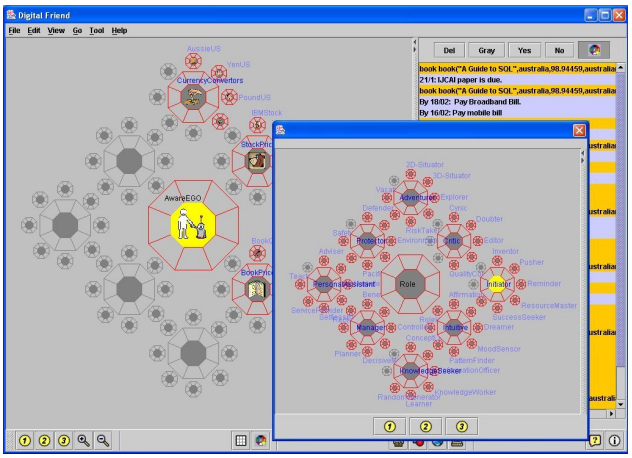
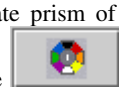


Figure 3. The role-based message lens

related to our formal and social responsibilities – i.e. they may be formal roles in the *social worlds* [14] we participate in by obligation and responsibility, such as *teacher*, *financial advisor, manager* – or lesser, more loosely recognized roles such as *inventor, trouble-shooter, critic*, or even fantasy sub-roles such as *explorer*. There is a design and development *methodology* for building a digital friend called the *Shadowboard Methodology*, which includes a list of 76 generic roles and sub-roles. Methodology [5, 3, 4].

The DigitalFriend is based on an agent *architecture* called the *ShadowBoard agent architecture*, which is itself based on notions from Analytical Psychology - specifically the *Theory of Sub-Selves*. While the theory is not important here, the *generic role-hierarchy* is, as it is drawn from the methodology and incorporated into the interface of the DigitalFriend implementation. The role hierarchy is used to view sub-sets of messages, divided according to these different roles - which are facets of the intricate prism of the individual's personality and life. By pressing the  button at the top of the *message stream* window (figure 1, right panel), we get the *role hierarchy dial* (note: it also uses the FUN interface), which lets the user select messages according to which role or sub-role in their life, they are currently focused on. The sub-agents get assigned to these roles during configuration.

3. A Simple Agent Type: Reminder Agents

Firstly, we look at the simplest agent *type* - Reminder Agents - to help get an appreciation of how the various types *integrate* into the overall system, before looking at more complex types.

Each cluster of agents, arranged around an octagonal tile in the agent interface, is called an *Envelope of Capability* (EoC). The EoC that is the focus of the example cluster in figure 4, is named *BillReminders.* It groups together a set of sub-agents, each of which has the task of reminding the user, when is the optimum time to pay a particular bill. The names alone tell us much of what we need to know in the example EoC: *BroadbandBill, Mobile-Phone-Bill, WaterBill, PaperDeadline*.

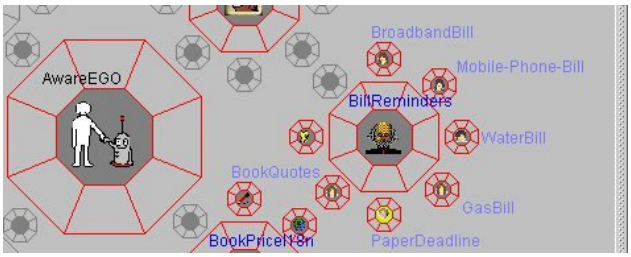


Figure 4. An *envelope* of reminder agents

By clicking on the *PaperDeadline* sub-agent and then selecting the *Modify/Current/Agent* menu sequence, the user generates the *Modify Reminder agent* dialog seen in the middle of figure 5. This dialog allows the user to modify the various *attributes* of the specific *PaperDeadline* reminder agent.

Some attributes are set when the agent is created: *name, role, sub-role, implementation*. The following attributes for a Reminder agent can be changed at any time by the user:

Attribute	**Description**
Start-Date-Time	*When to begin sending messages to the Message stream.*
Frequency	*How often to send the message.*
Unit-of-Frequency	*The unit of Frequency, either: seconds, minutes, hours* or *days.*
End-Date-Time	*When to stop sending messages to the Message stream.*
Alert-Message	*The text of the message being sent to the user.*
Auto-Stop-It	*Should the agent stop sending it after the End-Date-Time passes - True or False.*
Icon	*The visual face of agent as displayed within the DigitalFriend interface.*

Some of these attributes can be entered directly, while others use dialog windows for greater usability. E.g. *Start-date-time* and *End-date-time* both use a *rolling Calendar dialog,* seen in figure 5 below. Similarly, the *new* icon button leads to a file dialog window.

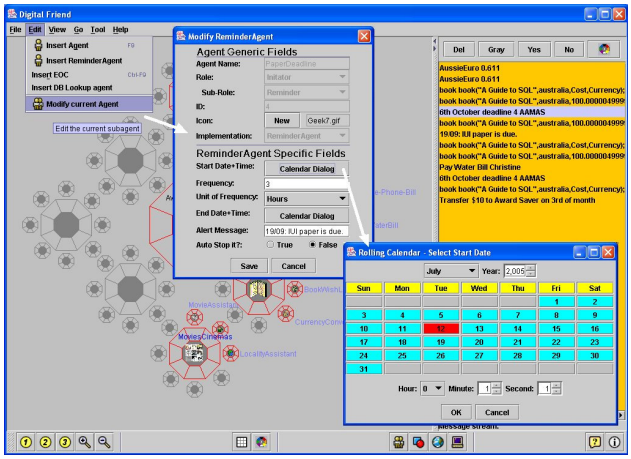


Figure 5. Modifying a Reminder Agent

Inserting a new sub-agent is similar to modifying an existing agent. First, an empty tile is selected (one that has a non-empty *parent* tile), and then the menu sequence *Edit / Insert ReminderAgent* brings up the same ReminderAgent dialog window seen in Figure 5, except this time the user creates all of the agent attributes.

4. Other Agent Types in the DigitalFriend

Reminder agents are simple to set up, however, by default they are also simple single-minded sub-agents, that don't entail any external monitoring or intelligence. The other agent types are more sophisticated.

Currently seven types of sub-agents are configurable within the DigitalFriend:

Agent Type	**Description**
ReminderAgent	*The simple but efficient reminder agent, as outlined above.*
WS Agent	*A web-service-wrapping agent, which retrieves information from the Internet periodically, via either the SOAP or WSDL open protocols.*
DB Lookup Agent	*An agent type that encompasses useful datastores which are kept within the DigitalFriend's Knowledge Tree (See Section 5).*
CoLoG EoC Agent	*The most flexible, intelligent and hence complex agent type. CoLoG is a constraint logic language [13], built into the DigitalFriend via an interpreter. These EoC agents have rules and goals and usually construct CoLoG programs dynamically - on the fly. They are usually brought into action by either their own sub-agents (bottom-up, data driven), when they in turn have some new data or message, or by agents higher up the hierarchy (top-down, goal-driven), in need of the specialist functionality of the CoLoG EoC Agent. See [3, 6].*
DB EoC Agent	*An EoC agent – in that it envelopes a group of sub-agents – but it also manages a datastore, into which some of its various sub-agents deposit information from time-to-time.*
DB SQL Agent	*Also an EoC agent – in that it envelopes a group of sub-agents – but in addition it manages a datastore or retrieves a dataset (local or remote), but one which uses embedded SQL language rather than CoLoG language.*
RSS Agent	*Periodically retrieves RSS feeds, in much the same way that the WS Agent retrieves web service information using SOAP - either timed, or prompt by another agent to do so.*
Native Java Agent	*Java programs which are able to communicate with other sub-agents, usually by computing new data and placing it in a parent DB EoC Agent.*

4.1 The WS Agent type

Next step up on the scale of sophistication after the Reminder Agents, is the WS (web-service) Agent type, which wraps a web service. It calls upon an external web service, usually somewhere out on the Internet or intranet, to provide it with external up-to-the-moment information of some sort, such as *currency conversion rates* and company *stock prices* of interest to a user – or any of thousands of possible web services that are available via the Internet. E.g. See the xmethods directory [16] for listings of SOAP and WSDL web services, that are commonly available.

Figure 6 above shows the dialog window for a WS Agent, in this

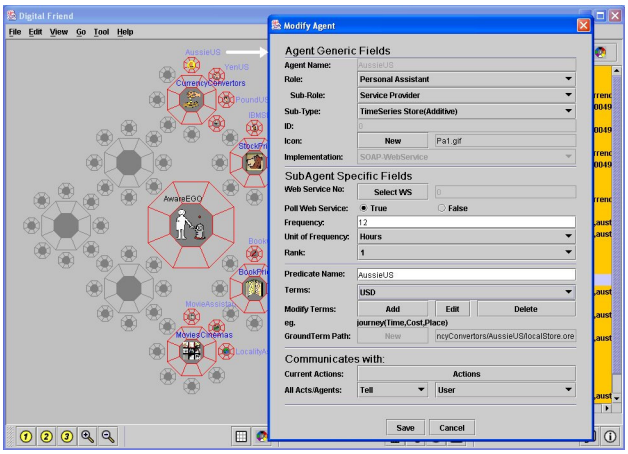


Figure 6. Modifying a WS Agent

case, one called *AussieUS*, which returns the conversion rate between the Australian and US dollars, at timed intervals.

Within the *Modify WS Agent* dialog are four visually distinct sub-panels, partitioned and named: *Agent Generic Fields*, *Agent Specific Fields*, *Predicate*-oriented fields, and *Communication* with other agents. Each encompasses a group of input fields and controls related to the partitioning:

4.1.1 Generic Fields:

The *Role* attribute currently chosen is *Personal Assistant*. The *sub-role* is *Service Provider* – these are from a set of role categories. Thereafter, messages that arise from this agent to the user, and arriving in the user's message stream, can be filtered by that role/sub-role using the *Role Message Lens* depicted earlier in Figure 3.

The *Sub-Types* field refers to three different styles of implementation of the WS Agent type, the first two are *Time-series store* (additive) and *Save-state Store* (replaces).

The datastore belonging to WS Agent *AussieUS* in Figure 6, can be perused by the user at any time, from the menu item sequence – *Tool / Ground Terms DB File*, or via a toolbar button. Figure 7 below shows the resulting window of US dollar conversions, each date-stamped. Time-series storing WS Agents are more likely to be used in *Decision Support System* (DSS) sub-agents, while *save current state* WS Agents are more likely to be used by real-time operational sub-agents.

Most WS Agents are of the second or third sub-type, which only store the *most recent* data item, and are often used in complex computations in smarter agents, such as in the example CoLoG EoC Agent outlined further down.

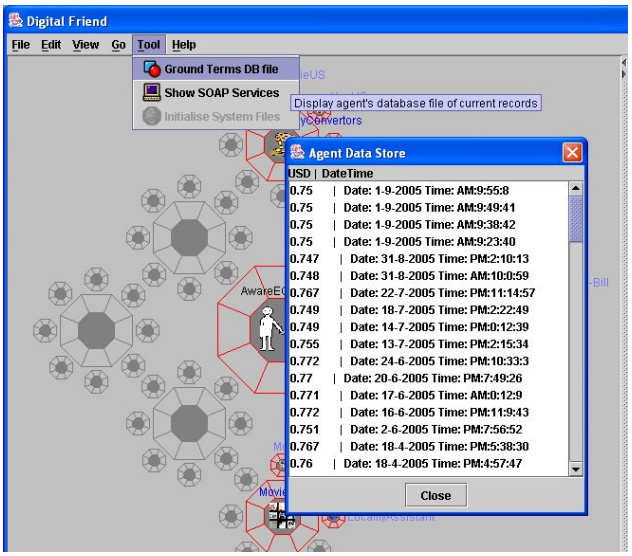


Figure 7. Time-series datastore for AussieUS WS Agent

Random browsing of the datastore by the user is not the primary designed use of most WS Agents, but it is possible to do so from within the GUI interface, for all agents that manage a datastore of some sort.

The following is a description of most of the other attributes of a WS Agent.

4.1.2 Agent Specific Fields (sub-panel):

Attribute	**Description**
Web Service No	*Which web service (that the DigitalFriend currently knows about) should be used by this agent. A Select-WS button leads to a user-friendly dialog listing many web services.*
Poll Web Service	A *true or false attribute that enacts the connection between the agent and the actual web service.*
Frequency	*How often should the agent poll the Web Service in search of fresher information.*
Unit of Frequency	*What is the unit of measure of Frequency – seconds, minutes, hours, days.*
Rank	*When there is an envelope of sub-agents that all do a similar task, but which call upon different web services to achieve the task, this field lets the user rank the quality or importance of the various competing web services.*

4.1.3 Predicate-oriented Fields (sub-panel):

Attribute	**Description**
Predicate Name	*The terminology used is nomenclature taken from computer logic languages – because a logic language called CoLoG, is built into the DigitalFriend.* ***Predicate*** *is the name from this nomenclature used for an entity about which there is a common set of related attributes.* ***AussieUS*** *is the example predicate name in figure 6 – the predicate name is usually the same as the name of the agent in the DigitalFriend.*

Terms	*A **term** is a name for a field or attribute in the logic language nomenclature. AussieUS(usd, time-stamp) – is the defining schema of the datastore for the example agent, where **usd** and **time-stamp** are the names of its two terms.*
Modify Terms	*Is a list of buttons that provide editing functionality to modify the naming and the number of **terms** in the list, i.e. the **Add**, **Edit** and **Delete** buttons.*
Ground-terms Path	*This is a file-path in the Knowledge Tree within the file space of the DigitalFriend. It leads to the actual file in which the datastore is kept. See the description of the Knowledge Tree further down.*

4.1.4 Communication Fields (sub-panel):

Attribute	**Description**
Current Actions	*This button leads to the list of existing inter-agent communication action paths, as listed earlier in Figure 2.*
All acts	*Leads to the list of all possible inter-agent communication acts, also called speech-acts.*
All agents	*Leads to a list of all agents currently in the DigitalFriend, with which the user/ developer may want the current agent to communicate.*

4.1.5 Configuring and Testing External SOAP Web Services

Independently of the web-service-wrapping agents that may use them, SOAP Web services can be *inserted*, *tested* and *modified* within a GUI interface in the DigitalFriend. In figure 8, the currently highlighted Web service is being *tested*. I.e. This particular Web service returns a numerical value of 1.915. The associated XML file *currency2.xml*, (termed, a *SOAP envelope*) asks the Web service at the URL http://services.xmethods.net for the exchange rate between the *US dollar* and *Pounds Sterling*.

The various sub-agents which can wrap a given web service, are setup to get the data they require, at timed intervals. The text-area at the top of the dialog, displays a green background indicating that this test was successful. On those occasions when the SOAP service does not correctly return a response from the web service, the background color of the text-area is red, and an error message is displayed.

Figure 9 below illustrates the fields within the DigitalFriend associated with a SOAP web service, accessible for editing after clicking the 'Modify' button shown in figure 8.

The *Show SOAP Service* dialog can be used to successfully configure, test and debug the interface to a SOAP web service, all from within the GUI interface.

Several WS Agents may all access the same web service in this list, usually doing so at different frequencies of access. I.e. One agent may need a fresh stock quote on IBM shares once per day, while a second agent may require fresh IBM shares quotes once per hour – the two agents call the same SOAP web service.

The GUI interface to web services, which is independent of the various agents, is accessed from the *Tool / Show Web Services* menu sequence. This level of indirection between web services

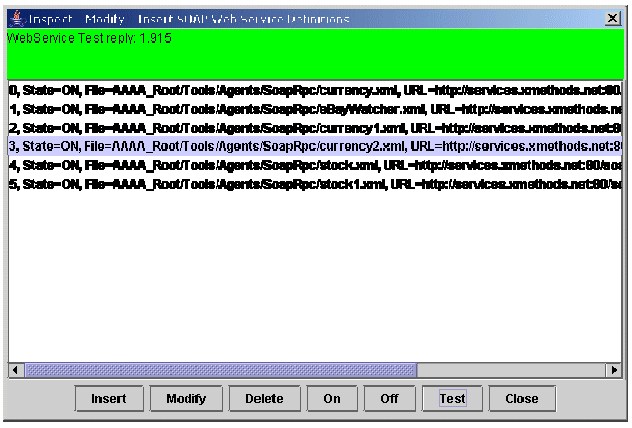


Figure 8. The Tool/Show-Soap-Services dialog interface

and the agents that call upon them, gives us considerable flexibility within the DigitalFriend. E.g. Multiple agents can make use of the same web service, at different frequencies.

Before looking at the other agent types it is advantageous to this overview, to first cover some of the more general features of the DigitalFriend, which now follow.

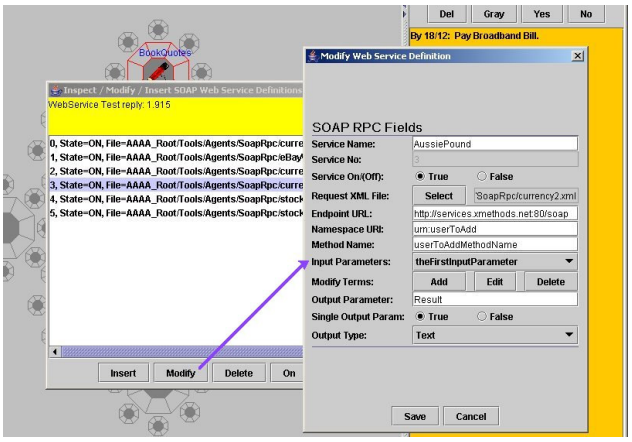


Figure 9. Modifying fields representing a SOAP web service

5. The Knowledge Tree with Implicit File Manager

The Knowledge Tree in the DigitalFriend has a built-in FUN-oriented file manager. It represents a sub-area of the client hard disk (or other secondary storage) and all the files in those folders that the DigitalFriend needs to deal with. The Knowledge Tree file manager displays *folders* (sub-directories) as octagonal tiles - it uses the same octagonal-tile FUN interface as already seen in both the *sub-agent hierarchy* (figure 1) and in the *Role hierarchy* (figure 3). Figure 10 shows the interface invoked by the *Knowledge Tree* menu item.

It displays the list of files in the currently highlighted folder (right-hand-side panel in Figure 10), any one of which can be selected for access to its file contents via the appropriate application package - current an image is displayed in Paint Shop Pro on Windows. On an Apple Mac it uses *Preview*.

Note: The DigitalFriend is cross-platform, running on Microsoft Windows, Mac OSX and Linux, as well other operating systems (OS) / platforms that support the standard Java language. A single configuration file is used to set the user's favorite applications for those three mentioned main platforms. It will also live-on and run-from a USB device, if the host machine has Java installed on it.

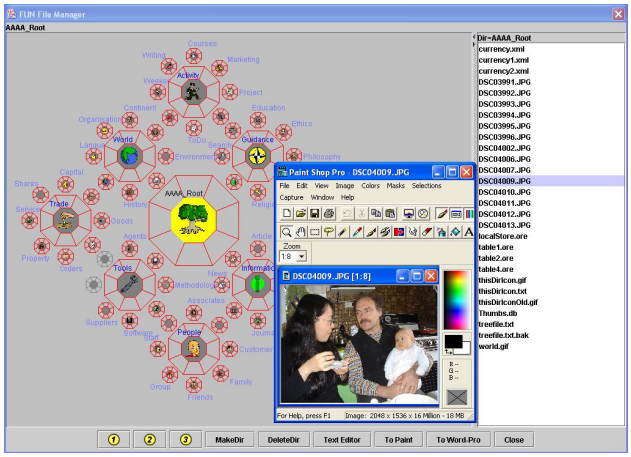


Figure 10. The File Manager displaying the *AAAA_Root* directory and files in it - the start of the *Knowledge Tree*.

5.1 The Knowledge Tree Ontology

The directory tree where all of the DB Lookup files, all of the *snippets* of CoLoG program (analogous to *sub-plans* in a BDI MAS), and any other files the user wishes, are stored within the installed directory path of the DigitalFriend on the host computer, at *DigitalFriend/classes/AAAA_Root*.

The Knowledge Tree begins at the directory ***AAAA_Root***, but after that, the choices of directory name are mostly flexible, and within the user's power to alter and expand. By default, there are several hundred sub-directories in there, many with default *icons* with appropriate symbolic value - this represents the default *ontology*.

The first exception to the *user's choice* statement above (i.e. which the user must not delete or rename) is the sub-directory ***AAAA_Root/Tools/Agents*** and all of its sub-directories.

The second exception pertains to the *maximum number* of sub-directories of any given directory in the Knowledge Tree (ie. beyond AAAA_Root), which is ***eight***. This is a usability design characteristic of the FUN interface, which has been researched and reported elsewhere [8,9].

The *default generic directory structure* which comes with the program, include names suitable for the sensible inclusive of many possible types of information. The first three-to-four generations are quite generic names (270+ sub-directories). This type of generic information structure can be called either an Ontology, a Taxonomy or a Knowledge Tree - depending on who you are talking to.

In addition to structure, the Knowledge Tree contains numerous DB lookup files of the example sub-agents supplied with the DigitalFriend, such as the *CountryCurrency.ore* datastore file represented in *Listing 2* further down.

An additional benefit of the DigitalFriend is gained for the user who uses multiple OS platforms during their normal day/ week/ month/ year, if they keep all of their work-a-day files within the AAAA_Root Knowledge Tree. They can move between a PC, a MAC and a LINUX desktop, and have exactly the same directory structure, and the same files on all systems, and use the different appropriate applications to access them on each platform. There is an option to *synchronize* content between these multiple instances of an individual's DigitalFriend, using a USB drive or similar ubiquitous storage.

6. The DB Lookup Agent type

An example file *currentCurrency.ore* which is stored in the Knowledge Tree at *AAAA_Root/World/* is a local database file, called a datastore. It has the following fields:

CountryCode	Internationally recognized two letter code.
Country	The name of the country in a particular record (row) of the file.
CurrencyCode	A three letter code for a given countries currency, internationally recognized.
Currency	The actual name of the currency e.g. *Australian Dollars*.

The contents of this datastore file are as follows (the first line defines the data fields of the database records that follow it):

Listing 1. **A generally static list of the world's 200+ countries, their codes and currencies**

```
countryCurrency(CountryCode,Country,CurrencyCode1,Currency):
countryCurrency(af,afghanistan,afn,afghani);
countryCurrency(al,albania,all,lek);
countryCurrency(dz,algeria,dzd,algerian_dinar);
countryCurrency(ad,andorra,eur,euro);
countryCurrency(ao,angola,aoa,kwanza);
countryCurrency(ai,anguilla,xcd,east_carribbean_dollar);
countryCurrency(ar,argentina,ars,argentine_peso);
countryCurrency(au,australia,aud,australian_dollar);
countryCurrency(at,austria,eur,euro);
countryCurrency(bs,bahamas,bsd,bahamian_dollar);
countryCurrency(bh,bahrain,bhd,bahraini_dinar);
countryCurrency(bd,bangladesh,bdt,taka);
countryCurrency(be,belgium,eur,euro);
... etc.
```

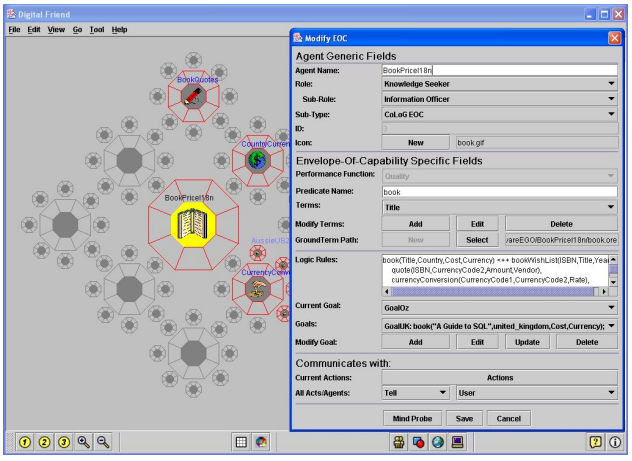


Figure 11. The Modify dialog for a CoLoG EoC agent

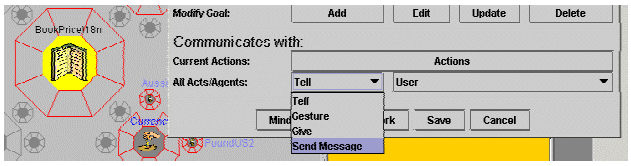


Figure 12. Sub-agents may use four types of speech-act

These relatively static, non-volatile datastores, which reside in the Knowledge Tree on the local computer, are so often useful in the DigitalFriend for their computational dexterity, that we introduced a specific agent type to deal with them - the *DB Lookup Agent*. Each DB Lookup agent is effectively a guardian of the underlying local datastore.

7. The CoLoG EoC Agent Type

CoLoG is a constraint-logic language, enacted within the DigitalFriend available as an interpreter at runtime. These agents are EoC agents – meaning that they *manage* a group of sub-agents, which can be recursively deep. CoLoG EoC agents are the most sophisticated, hence the most flexible and useful agent type in the DigtialFriend MAS. Figure 11 shows the working *Modify* dialog box for a CoLoG EoC agent.

7.1 Logic Rules and Goals (sub-panel):

Space does not suffice to describe in detail all of the attributes of a CoLoG EoC Agent, so this is a brief description of the attributes of most interest, or most different from those covered in other agent types described earlier.

Attribute	**Description**
Logic rules	*This field holds logic rules expressed in the CoLoG language (Prolog-like), which bind together the various sub-agents in the immediate EoC. The firing of these rules in the running system, can be data-driven from bottom-up - e.g. from child web services receiving new data, or goal-driven from above in the agent hierarchy.*
Goals	*Goals are predicates with some of their terms specified as variable/unknown. The system provides the missing elements. I.e. A goal is another name for a query. An agent may store any number of predefined goals, any of which can be called upon by other agents.*
Modify Goal	*A set of controls for GUI editing of goals.*
Mind Probe	*A button that leads to an interactive GUI interface with the CoLoG language, but with the current state of the dynamically generated program listed. This lets the user test and run any of the EoC Agents programs, and is useful in debugging logic rules, and web service and other information feeds, in the DigitalFriend.*

CoLoG EoC Agents provide the glue that orchestrates multiple sub-agents - including multiple web-service wrapping agents - into new functionality. The CoLoG programs are constructed on the fly, from sub-agents and web service information. The other critical component that makes the DigitalFriend configurable by an *end-user*, is the inter-agent communication language, accessible from the GUI environment.

8. The Inter-agent communication Language

Although the various sub-agents that together make up a digital friend are structured into a hierarchy, they need to be able to *communicate* to fellow agents that may be on the same or on other branch in the hierarchy. I.e. With respect to inter-agent *interactions*, a simple hierarchy would be quite restrictive. To facilitate inter-agent communications, we found that a minimum of four *actions* (also known as *speech-acts*) needed to be incorporated into the built-in CoLoG language, to give the needed flexibility to build generic systems using the DigitalFriend tool.

The following lists the four actions and what they tell the underlying software to do (in all cases, there are two sub-agents involved in a speech-act – a *sender* and a *receiver*):

Action-command	**What it does.**
Tell	*Indicates to the receiver agent, that something has already been done by the sender agent.*
Gesture	*Indicates to the receiver, that there is some state-change in the sender.*
Give	*Passes some data (predicate ground-terms/records or objects) from the sender to the receiver.*
Send Message	*Sender agent sends a textural message to the receiver.*

Figure 12 above shows a combo-box control, within an '*Edit Agent*' sub-dialog that allows the user to select one of the four communicative-actions.

The difference between a *Gesture* and a *Tell* communicative-action, is that the *Gesture* indicates a change in the sender, so that the receiver may or may not choose to take some action based on it; while the use of *Tell* indicates to the receiver, that some new action that affects the receiver has *already been done*, and the receiver may or may not take some further action.

Figure 2 earlier on, shows all the *current* communicative-actions within the DigitalFriend, for the current starting set of example sub-agents supplied with the tool.

The *speech-acts* are entered and edited via the *Modify Agent* dialogs. At the bottom of the Modify EOC Agent dialog in Figure 13, there is a sub-dialog with the label '*Communicates with:*' which has a button labeled *Actions*. Pressing the *Actions* button, followed by the other user interactions, leads to the cascading series of dialogs seen in Figure 13 below.

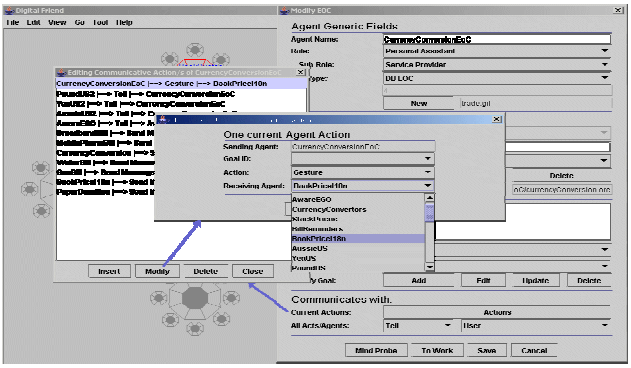


Figure 13. Insert/Modify dialog to enter speech-acts

An extensive example of an EoC CoLoG agent that orchestrates several external web services, and several DB Lookup agents, into a new more powerful agent, one probably not envisaged by the providers of the individual web services, is fully detailed in [5].

9. Open Source Developer Opportunities

The DigitalFriend is the first implementation of the ShadowBoard agent architecture (a blueprint) - which may be characterized as a *Multi-Agent System* (MAS). It fulfills much of the promise and more, outlined in the blueprint in Goschnick [3]. While the original concept foresaw the use of numerous orchestrated sub-agents bound together with a runtime logic language, *web services* have instead come to be the main source of such external functionality, nonetheless they are bound together at runtime with the logic language, as called for in the original design. Similarly, RSS feeds have now been added as sources of external intelligence and information update.

The code has been open sourced under a combination of GPL and LGPL license agreements. As more of the code is tidied up and better documented, more of it will be moved into the GPL licensed section. End-users and developers alike, can add to the functionality of their DigitalFriend in at least four way:

1. Write original sub-agents *in Java*. The *method* signature of a class, is interpreted as a logic predicate in the internal CoLoG, enabling any Java program to be integrated directly into one's DigitalFriend
2. Write mash-up scripts *in the CoLoG language*, as these are interpreted at runtime, via the internal CoLoG language interpreter.
3. Add sub-agents that wrap SQL *Select* commands, giving access to any application written *in a relational DBMS*.
4. Write a purpose-built *Web service* (or a specifically tailored RSS feed), and then combine it into a *mash-up* in your DigitalFriend, with various sub-agents in Java, SQL and CoLoG.

This end-user-oriented desktop tool is extremely open-ended, and time spent downloading, using and extending it, will be richly rewarded in personal productivity. In addition to the programming opportunities outlined above, as an everyday user tool it has: some implicit file manager functionality; it can synchronise user files and folder structure between Windows, Mac OSX and Linux computers, via USB and other portable devices (e.g iPod); and it has a RSS feed *editor* built-in. It can also be run from the USB device itself, if the host system has Java installed.

10. ACKNOWLEDGMENTS

The software tool *DigitalFriend V1.0*, is a product of a development effort titled *The Digital Self Project*, funded by a Telstra Broadband Fund Development Grant, by Telstra Ltd. The DigitalFriend product is the first software tool from the effort to become available for public use, downloadable from www.DigitalFriend.org as of early December 2006.

11. REFERENCES

[1] Bradshaw, J. *Software Agents*. MIT Press, 1997.

[2] Goschnick, S.B. Enacting an agent-based digital self in a 24x7 web services world. In *Proceedings of ISMIS 2003, the 14th Symposium on Methodologies for Intelligent Systems*, Springer LNAI 2871, pp 187–196, Japan, 2003.

[3] Goschnick, S.B. *ShadowBoard: an Agent Architecture for enabling a sophisticated Digital Self*. Thesis, Dept. of Computer Science, University of Melbourne, Australia, 199 pages, 2001. Available at: eprints.unimelb.edu.au

[4] Goschnick S.B. ShadowBoard: A Whole-Agent Architecture that draws Abstractions from Analytical Psychology. In *Proceedings of PRIMA 2000*, August, Melbourne, Australia, 2000.

[5] Goschnick, S.B. & Graham, C. Augmenting Interaction and Cognition using Agent Architectures and Technology Inspired by Psychology and Social Worlds. *Universal Access in the Information Society*, 4(3), Springer, pp.204-222, 2006

[6] Goschnick, S.B. and Sterling, L. Enacting and Interacting with an Agent-based Digital Self in a 24x7 Web Services World. In *Proceedings, IEEE joint conference on Web Intelligence and Intelligent Agent Technology* (WI/IAT), Halifax, 2003.

[7] Goschnick, S.B. & Sterling, L. Psychology-based Agent Architecture for Whole-of-user Interface to the Web, *Proc. of HF2002 Human Factors Conference: Design for the Whole Person - Integrating Physical, Cognitive and Social Aspects*, Melbourne, November, 2002.

[8] Lane, S. *User Interfaces for Navigating Information Hierarchies*. Honours thesis, 68 pages, October 2004.

[9] Lane, S., Goschnick, S.B. and Smith, W. Evaluating the FUN Interface, submitted to: IUI-2006, Sydney 2006. www.dis.unimelb.edu.au/staff/gosh/EvaluateFun.pdf

[10] Libby, D. (1999) *RSS 0.91 Specification*. http://my.netscape.com/publish/formats/rss-spec-0.91.html. Cited 2006.

[11] Rao, S. A. and Georgeff, M.P. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[12] SOAP Version 1.2 Part 0: Primer W3C Candidate Recommendation, http://www.w3.org/TR/soap12-part0/ Cited, 2006.

[13] Sterling, L. and Shapiro, E. *The Art of Prolog*, Second Edition, The MIT Press, Cambridge, USA, 1994.

[14] Strauss, A. A Social World Perspective. *Studies in Symbolic Interaction*, Vol 1, 119-128, 1978.

[15] *Web Services Description Language (WSDL) Version 1.2 W3C Draft*, http://www.w3.org/TR/wsdl12/ Cited 2006

[16] www.xmethods.net. Cited 2006.