Agent-based Integration of General-Purpose Tools

Stephen Cranefield⋆ and Martin Purvis†
Department of Information Science
University of Otago
P.O. Box 56, Dunedin, New Zealand
⋆ scranefield@commerce.otago.ac.nz
Phone: +64 3 479 8083
Fax: +64 3 479 8311
† mpurvis@commerce.otago.ac.nz
Phone: +64 3 479 8318
Fax: +64 3 479 8311

Abstract

Agent-Based Software Integration (ABSI) entails the development of intelligent software agents and knowledge-sharing protocols that enhance interoperability of multiple software packages. Although some past ABSI projects reported in the literature have been concerned with the integration of relatively large software frameworks from separate engineering disciplines, the discussion in this paper concerns the integration of general-purpose software utilities and hand-crafted tools. With such smaller-scale ABSI projects, it may be difficult to justify the expense of constructing an overall ontology for the application. There are cases, however, when the project involves general-purpose tools that manipulate the same general entity types (such as files) but at different levels of abstraction. In such cases it is appropriate to have ontologies appropriate for the general usage of each tool and constraint descriptions that enable the ontological specifications to be mapped across the various levels of abstraction. This paper discusses issues associated with this type of ABSI project and describes an example information management application associated with university course administration. For the information management application presented the key issues are the provision of standard agent wrappers for standard desktop information management tools and the design of standard ontologies describing information stored in relational databases as well as in structured text files. Examples of a conceptual model describing such a database ontology are presented in connection with the example application. It also suggested that a general planning agent, distinct from the notion of a facilitator agent, be employed in this context to assist in the use of various agents to manipulate information and move items from one data format to another.

Keywords: Software interoperability; software agents; ontologies, knowledge-sharing; KQML

Agent-based Integration of General-Purpose Tools

Abstract

Agent-Based Software Integration (ABSI) entails the development of intelligent software agents and knowledge-sharing protocols that enhance interoperability of multiple software packages. Although some past ABSI projects reported in the literature have been concerned with the integration of relatively large software frameworks from separate engineering disciplines, the discussion in this paper concerns the integration of general-purpose software utilities and hand-crafted tools. With such smaller-scale ABSI projects, it may be difficult to justify the expense of constructing an overall ontology for the application. There are cases, however, when the project involves general-purpose tools that manipulate the same general entity types (such as files) but at different levels of abstraction. In such cases it is appropriate to have ontologies appropriate for the general usage of each tool and constraint descriptions that enable the ontological specifications to be mapped across the various levels of abstraction. This paper discusses issues associated with this type of ABSI project and describes an example information management application associated with university course administration. For the information management application presented the key issues are the provision of standard agent wrappers for standard desktop information management tools and the design of standard ontologies describing information stored in relational databases as well as in structured text files. Examples of a conceptual model describing such a database ontology are presented in connection with the example application. It also suggested that a general planning agent, distinct from the notion of a facilitator agent, be employed in this context to assist in the use of various agents to manipulate information and move items from one data format to another.

1 Introduction

In today's heterogeneous software environments, with tools written at different times for various specific purposes, there is an increasing demand for interoperability among these tools [1]. The goal is to combine information and services from individual tools in order to solve problems or carry out tasks that could not be accomplished by a single tool. Progress in this area has been assisted by the development of the notion of intelligent software agents, which can model both tools and people and which communicate by means of formal information sharing protocols known as *ontologies* [2]. "Agent-Based Software Integration" (ABSI) [3] is the expression used to characterise the development of agents and ontologies that assist in the enhancement of software interoperability. Notable examples of efforts in this area are the PACT [4] and SHADE [5] projects in collaborative and concurrent engineering that seek to integrate large multitool frameworks, each of which was developed for a specific engineering discipline. The focus of this paper, however, is concerned with a smaller scale of project that may involve the multiple use of general-purpose utilities and individually constructed application programs. In such cases the computer user wishes to solve a particular problem by using tools, some of which may be "hand-crafted", that exist at various levels of generality. Because of the changing nature of these tasks, the time and resources to build an integrated framework are usually not available. For example in an exploratory investigation, a researcher may attempt to use any of a number of small tools for analysis or visualization that were developed for a specific problem in a particular programming environment. Although efforts have been made to integrate such tools into large multitool frameworks or workbenches [6], the nature of exploratory investigation is such that there will always be a need for the more open-ended approach of stitching together newly-made investigative tools. For more "routine" administrative tasks, the same conditions can apply. Here we discuss issues relating to the construction of software interoperability agents, which we might call "desktop agents", associated with the management of information in a heterogeneous computing environment.

Both the PACT and SHADE projects are part of the ARPA Knowledge Sharing Effort, which seeks to develop a common agent communication language (ACL) that will facilitate knowledge sharing among agents across heterogeneous platforms. ACL has three components: a core language, KIF (Knowledge Interchange Format) based on first order logic; an interaction language for expressing messages, KQML (Knowledge Query and Manipulation Language) which defines "performatives" that are based on speech act theory [7]; and a vocabulary in which to express agent ontologies. ABSI projects using ACL discussed in the literature have to date largely focused on the domains where the software tools to be integrated are complex and/or expensive to develop, *e.g.* concurrent engineering of a robotic system [4], civil engineering [8], electronic commerce [9], and distributed civic information systems [10]. In these domains the time and cost of encapsulating an existing software framework as a KQML-speaking agent is small compared to the cost or complexity of that framework and benefits of enhancing the interoperability of a complete suite of tools. Moreover time spent in designing appropriate ontologies for the given specific problem domain is likely to be worthwhile due to the large investment in the overall system.

In contrast with these large-scale integration efforts, this paper considers issues relating to an example of ABSI in connection with day-to-day computer-assisted administrative tasks performed by many workers to support their principle occupation (which may fundamentally involve computers). To meet their administrative requirements, such users are likely to develop a basic familiarity with a number of different software packages and to build up associated usage patterns. When the requirements of the job change or new software becomes available, the old methods may no longer be appropriate for these routine tasks. However, it may be difficult to justify (or event to find) the time to learn how to use new tools and usage patterns. While it may be possible to automate standard patterns of interoperation of various software tools (*e.g.* by using scripting languages), for routine administrative tasks this may be considered to be too time-consuming (at least in the short term) to be worthwhile. Thus we may consider the idea of ABSI to enhance the interoperability of the tools and the maintainability of established work patterns. However, as is discussed below, the construction of a single problem-specific ontology in connection with the use of such general-purpose tools may not be appropriate.

An example problem domain which will be used to illustrate the issues discussed in the paper is the administration of a university course. This task is currently performed by the first author using a variety of different tools and systems: the central university records system (running in-house software under VMS), a dBASE course-specific database application, a Visual Basic program for marking electronically submitted assignments on-line, a DOS program allowing students to view their marks on-line, Microsoft Excel (for preparing graphs and summaries of student marks) and various Unix, DOS and Novell Netware utilities. Some of these tools existed independently of this course, others were designed specifically for it, and some were developed hastily to fill an immediate need. Information is exchanged between these tools in the form of text files and the required file format conversions are performed using the tools the course administrator is most familiar and comfortable with: Unix utilities such as awk and join, and the editor emacs.

This situation is probably typical of many routine computer-supported administrative tasks:

- Some pre-existing software tools must be used.
- Some special-purpose tools are used (often hastily written with no frills).
- The task could be performed more efficiently by using other (possibly newer) tools with better support for interoperation, and/or by automating parts of the process.
- Successful performance of the task requires the user to remember the correct sequence of actions to perform on the various tools, and to perform these without making mistakes.

Tasks such as these could benefit greatly from ABSI technology provided that the overheads of "agentifying" the task could be kept small enough for the user to consider the time investment worthwhile. This paper considers what support a user would need in order to make desktop software interoperability using an agent-based approach viable. In particular, two needs are identified:

1. The provision of "agent wrappers" for standard desktop tools
2. The design of standard ontologies describing information stored in relational databases as well as in structured text files

1.1 Desktop agents

The vast majority of computer-supported work is probably performed using a relatively small number of common desktop products, *e.g.* the market has only left a few competing word processors, spreadsheets, and database systems. Most of these include extension languages or communication protocols that would allow standard 'wrapper code' or 'transducer' applications to be developed to encapsulate particular uses of these applications as agents with their required inputs and resulting outputs described declaratively in KIF. For example, one agent might encapsulate the ability of the Excel spreadsheet to take as input a text file containing a column of numbers and produce a printed histogram of these numbers. In general, one agent wrapper could be provided for each application, and this would register with a special planning agent all the functionalities of the underlying application that have been encoded in the agent.

While it is not envisaged that the full functionality of general packages such as spreadsheet or word processing applications could be expressed declaratively in a single specification, it should be possible to provide specifications for various different actions that the application can perform. If the agent wrapper is designed to be easily extensible then over time other uses of the application could be included in the agent's capabilities by plugging in modules provided by third parties.

1.2 Standard Ontologies

Researchers at the Stanford University Knowledge Systems Laboratory maintain a library of ontologies that have been developed for various domains [11]. The ontologies currently available either describe the domains of particular ABSI projects such as concurrent engineering, or are general utility theories (*e.g.* describing sets and lists) that can be included when developing new ontologies. For example, one particularly useful ontology is the frame ontology (named after the AI knowledge representation structure of the same name). This defines a terminology that can be used to define a domain in an object-oriented fashion.

When integrating specialist software tools that are used in a particular application area, agent-based software integration involves designing an ontology for that problem domain and encapsulating the various tools as agents that communicate using that ontology. In other words, the agent-encapsulated tools communicate using a common high-level ontology[1].

For routine tasks performed using standard software tools, however, it is not practical to express the particular modes of use made of these tools in terms of one particular problem domain. The great value of many desktop tools is their general usefulness for many day-to-day tasks. To ensure maximum reusability, an agent encapsulation of such tools should describe their capabilities in the most general terms possible. This implies, for example, that a tool that deals with text files needs to be described in terms of an ontology of text file formats. An agent encapsulating some of the functions of a database utility should communicate using an ontology describing tuples and relations. Unfortunately, at present there seen to be no ontologies available to describe these low-level but generic data representations. Section 4 will present some initial ideas about how such ontologies should be structured.

2 System architecture

Figure 1 shows the proposed architecture for a desktop agent system. The user will interact with a console agent that will accept requests expressed using the ontology of the user's various problem domains (preferably via a graphical user interface rather than requiring the user to enter KQML requests directly). These requests will be sent to a facilitator agent which will route the messages to the appropriate agent based on its knowledge of the capabilities of the agents registered with it (this is known as "content-based routing" [1]).

The facilitator agent developed by the Stanford University Computer Science Department's Logic Group [3] performs content-based routing by requiring agents to send it facts of the form:

$$\texttt{handles}(\textit{agent},\ \textit{performative-schema})$$

Whenever the facilitator receives a request *performative-instance* (*i.e.* a performative with some arguments), it attempts to prove (using backward chaining) a fact of the form

$$\texttt{handles}(\textit{some-agent},\ \textit{performative-instance})$$

[1] In some cases the tools may have different but related ontologies, but these are still high level problem domain oriented ontologies and messages can relatively easily be translated from one ontology to another.

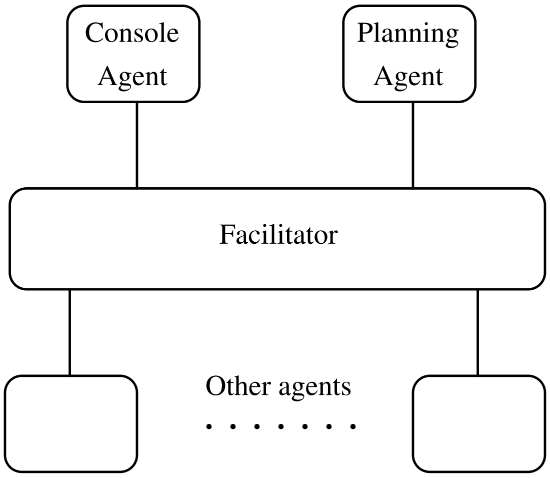

Figure 1: A desktop agent system architecture

This inference capability allows agents to specify that they can handle actions conditionally (*e.g.* a print manager agent that can print files provided they are smaller than one megabyte in size [12]). Agents can also send the facilitator new clauses that extend its abilities to perform content-based routing (see [12] for an example).

For interoperating systems involving a few complex agents the patterns of interaction between the agents are likely to be known in advance and limited in scope. In contrast, for desktop agent systems, there may be many simple agents each encapsulating a number of operations performed by various general-purpose tools. To achieve a user's goal, it may be necessary for a number of different agents to perform actions in an appropriate sequence. It would be tedious if the user were required to specify this sequence of agent invocations. However, deducing an appropriate sequence of agent actions to achieve a goal is an AI planning problem and the currently available facilitators do not support planning well. In particular, facilitators reason over a knowledge base expressed using the knowledge representation language KIF which has no notion of state or time. While it would be possible to write KIF rules that allowed the Stanford facilitator to perform planning, this seems to be stretching the role of a facilitator too far. It would be more modular and efficient to provide a specialised planning agent.

The planning agent would inform the facilitator of its ability to handle performatives matching the schema `(achieve ?goal)`. Each agent would need to send the planning agent STRIPS-style specifications of the actions it can perform (consisting of the action's preconditions and add and delete lists).

3 An example application

To illustrate the operation of a desktop agent system we will consider the integration of two programs used in the university course administration domain discussed in Section 1[2]. For this

[2] For the sake of brevity some details of this domain will be simplified, *i.e.* it will be described as being more easily integrated than it really is!

Table	STUDENT		
Attribute	StuID	StuName	NW_ID
Domain	String	String	String

Table	CMPT			
Attribute	CmptID	Descript	OutOf	Weight
Domain	String	String	Number	Number

Table	ASSESS		
Attribute	StuID	CmptID	Mark
Domain	String	String	Number

Figure 2: The base tables of the course database. STUDENT records student details, CMPT records assessment components and ASSESS holds students' marks for each assessment component.

---

course, the students' C++ programming assignments are submitted electronically and stored on the network in directories indexed by the students' Novell Netware ID. To assess the programs a simple Visual Basic marking utility is used. This displays a list of student names beside a column of buttons. Clicking on the button beside a student's name finds the appropriate network directory and starts up an instance of the Turbo C++ application with the student's project file opened. The marking utility also has a column of text boxes where the students' marks can be entered. The marking utility reads in a text file containing (amongst other details) student names and Netware IDs and when the marking process is finished the file is written out with the students marks' added. Once an assignment is marked, the marks must be entered into the dBASE database system used to record the student details. A simplified version of the relational data model implemented by this database is shown in Figure 2.

For any given assignment, in order to create the input file for the marking utility, mark the assignments and record the marks in the database the following steps must be performed:

1. Access the STUDENT table in the database and generate a text file containing student ID numbers and names (separated by commas, with each student's record on a new line).

2. Run the marking utility. When the user quits this application the input text file will be written out with a new field containing a mark at the end of each line (with a comma preceding it).

3. Process the file output by the marking utility to generate a comma-delimited text file with each line containing a student ID, the component ID (the name of the current assignment) and the mark.

4. Add the information from this file to the table ASSESS in the database.

This process is illustrated in Figure 3.

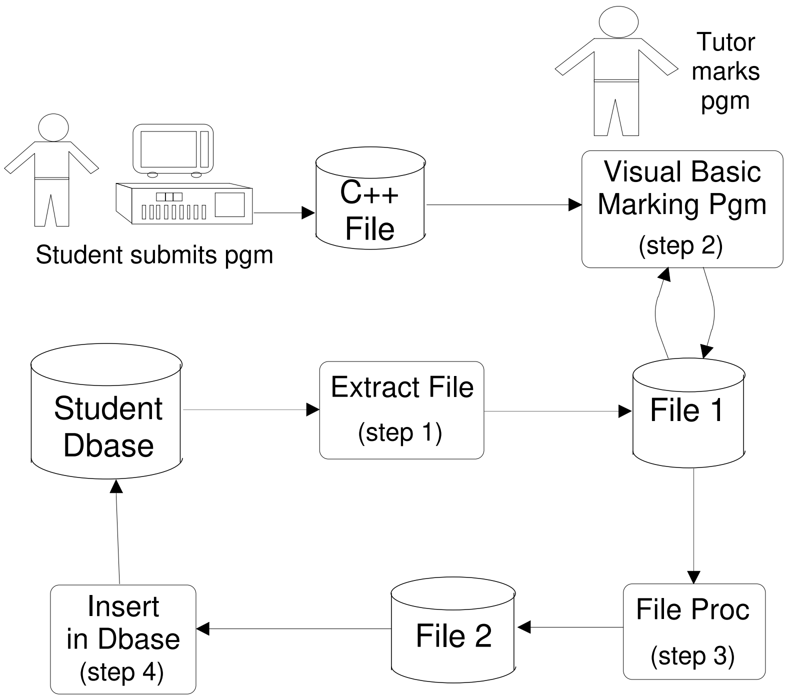

Figure 3: The assignment marking process

4 Ontological issues

4.1 Relational data models

In any ABSI project there needs to be at least one high-level ontology describing the problem domain (in some cases there may be several due to different agents having different views of the domain). In the domain described above, the problem area involves student details, assignments and marks. An ontology for this domain could be created in various ways, *e.g.* an object-oriented model could be based on the Stanford ontology library's frame ontology. However, like all applications involving a relational database, a domain model has already been developed: the relational data model developed for the database application. This could be easily described as an ontology if a standard ontology of relational data models were available. Such an ontology would define relations in terms of their attributes and attribute domains and would need a concept of base relations as well as a way of naming relations derived from the base relations. The discussion in this paper assumes the existence of such an ontology with relations described using the relational algebra [13], *i.e.* relations can be built from the base relations using operations such as select, join and project.

For the course administration domain, using a domain ontology based on a relational data model, a tuple in the base relation STUDENT (see Figure 2) might be represented by the following KIF fact:

```
(tuple (setof (stuid "9501234")
              (stuname "Joe Smith")
              (nw_id "A0100001")   ) student info202-dm)
```

where `student` names the relation and `info202-dm` names the particular relational data model (there could be more than one data model defined). Note that although the specification of KIF does not include characters, strings and other standard data types, for ease of discussion we will assume the use of an extended version that does.

4.2 File formats

In ABSI projects where there are a fixed number of interoperating tools that are specialised for a particular domain, it makes good sense to encapsulate each tool within an agent that speaks a high-level domain-related ontology. In contrast, in the type of desktop agent system proposed in this paper, there may be agents controlling various general-purpose tools that act at a relatively low data representation level (*e.g.* by performing manipulations on files). As these tools could be used in various problem domains it would limit their reusability in an ABSI framework if their agent wrappers could only declare their abilities in terms of a single high-level domain. Although for a general purpose tool it would be possible to generate separate agent wrappers corresponding to different domain ontologies, this would clearly be a duplication of effort. Instead, a generic tool should be described at the level at which it operates. Thus a utility that can manipulate text files should be described in terms of an ontology of text files.

There are a number of different levels at which the contents of files are traditionally viewed. Figure 4 shows a hierarchy of possible ontologies that could be constructed for representing the contents of a text file in a declarative fashion, along with the type of facts that be would be used to describe the file contents at each level. To relate the different viewpoints the ontologies would also need to describe how to relate the information at the different representational levels. For instance, the information that the file "students.dat" is a text file with three string-valued fields delimited by commas might be represented by the following fact (in the "extended KIF" we use for convenience in this paper):

```
(file "students.dat" (delimited #\, (list-of string string string)))
```

This predicate would be defined in an ontology by a formula inferring facts about records and fields from facts about facts about lines in files. For instance, the fact

```
(line 10 "students.dat" "9501234,Joe Smith,A0100001")
```

would allow the following three facts to be deduced:

```
(field 1 10 "students.dat" "9501234")
(field 2 10 "students.dat" "Joe Smith")
(field 3 10 "students.dat" "A0100001")
```

Thus, the `file` fact above provides the information needed to translate between the line-based view of a file and the record-based view. Similarly, facts of the following form could be used to link the record-based and the relational views of a file:

```
(file-represents-relation ?file ?rel ?data-model ?att-list)
```

Conceptual Level	**Fact schema**
Relational	`(tuple ?pairs ?rel ?dm)` *pairs* is the set of attribute-value pairs in relation *rel* of data model *dm*

$\Uparrow$

An ontology relating lists of records to sets of tuples, using facts asserting which relation and data model a file corresponds to and which fields correspond to which attributes.

$\Downarrow$

Records and fields	`(field ?i ?n ?f ?v)` Field *i* of record *n* in file *f* has value *v*

$\Uparrow$

An ontology for string parsing, based on facts describing the record format.

$\Downarrow$

Seq. of lines	`(line ?n ?f ?s)` Line *n* of file *f* is the string *s*

$\Uparrow$

An ontology describing newline terminated strings.

$\Downarrow$

Seq. of chars	`(char ?n ?f ?c)` Character *n* in file *f* is *c*

Figure 4: A hierarchy of text file ontologies

As tuples in relations are *sets* of attribute-value pairs, whereas the fields in a record are ordered, the final argument `?att-list` is required to specify the order in which the attributes appear in the file — this is a list which is some permutation of the attributes of the relation `?rel`.

It is important to note that ontologies are only a *specification* of a standard terminology for a domain. While some agents may perform inference using the formulae in an ontology, all that is *required* of an agent is that its use of the terminology in an ontology be consistent with the formulae in the ontology. Defining ontologies describing different conceptual views of files, and the way to translate between them, does not imply that all file processing will be performed in an inference system. In practice, these ontologies — particularly at the lower levels — might be implemented by 'procedural attachment', so that telling a file manager agent a `file` fact such as the one above would result in that agent sending a stream of `field` facts to all interested agents.

4.3 A database ontology

The relational data model ontology discussed in Section 4.1 describes a conceptual model of a domain, whereas the text file ontologies discussed in the previous section describe the physical representations of data. However, the principal means of representing relational data is not using files but rather (more sensibly) using a relational database. Just as the text file ontologies refer to files, we need an ontology in which there is a concept of a database (as opposed to the conceptual model implemented by it). A database is a separate concept from a relational data model: a database could represent information from several relational models; conversely the information pertaining to a single data model could be split (or duplicated) across several databases.

In the discussion that follows we will assume the existence of a database ontology which defines (amongst others) facts of the following form:

```
(database-matches-datamodel ?database ?data-model)
```

and

```
(foreign-key-ok ?key ?table ?database)
```

The first of these declares that the tables in the database match the relational data model specified. The second states that the value `?key` appears as a key in table `?table` of the database (*i.e.* it represents an integrity check). These would be implemented by procedural attachment to the database query language.

5 Agent action specifications

In this section we assume the existence of ontologies for relational data models, text files and databases as sketched out above. Using the terminology of these ontologies it is now possible to express the input requirements and output results of each of the four steps of the assignment marking process. As argued in Section 2, representing and reasoning about the capabilities of agents is beyond the capabilities of current facilitators (and is a specialised task that probably does not belong in a facilitator anyway). Instead, our desktop agent architecture includes a planning agent. It is proposed that for each action that an agent can perform, it sends a message to the planning agent describing that action as a parameterised planning operator — with its pre- and postconditions expressed in KIF using the lowest-level (and therefore most general) ontology possible. Figure 5 shows what the action specifications might look like for steps 2 and 4 of the assignment marking process. These action specifications could be used by a planning agent to plan a sequence of agent actions to achieve a goal.

Note that both the `mark` and `add-marks` actions in the figure are inherently related to the course administration domain, so their specifications include statements at the text file level (the actions require text files as inputs) as well as at the relational level (these declare that the files represent particular relations in the data model for the problem domain). However, not all the agent actions performed in a desktop agent system will be specialised to a particular problem domain in this fashion. Consider step 3 of the assignment marking process. This is a text file manipulation process that takes the file produced as a result of running the marking program and converts it into a format suitable for adding information to the ASSESS table in the database. This can be performed by two operations at the text file level: deleting two fields of the file and then adding a new field containing the assignment name in each row. Figure 6 shows how these two file-level operations can be used to achieve the preconditions of the `add-marks` action starting with the postconditions of the `mark` action.

To infer that this sequence of file manipulations can be used as a link in a plan involving higher level concepts would require the planner to drop down a level in the hierarchy of ontologies, generate a subplan at that level and then produce a specification of that subplan at the higher conceptual level. It is not yet clear how this would be done. Hierarchical planning techniques may be useful here, but it may also involve inferences that go beyond classical planning techniques.

6 The planned implementation

Currently there seems to be no KQML application programmers interface and facilitator that runs under Windows (although there are plans to port the Stanford C++ KQML API to Windows with it running as an OLE server [14]). In the meantime, it is planned to implement a desktop agent system for the course marking domain using Amzi! Prolog + Logic Solver [15]. The Prolog listener will act as console agent as well as filling the roles of the planning agent and facilitator. The ontologies will be defined as a Prolog rule base. Agents will be directly invoked from the Prolog session using Windows API and other DLL calls defined as Amzi! Prolog external predicates.

7 Conclusion

Enhanced software interoperability of multiple general-purpose and/or hand-crafted utilities may be brought about by means of ABSI. In order to retain optimal benefits of the tools and utilities, it is best to maintain ontological descriptions at the appropriate level of the tool. With ontological descriptions at multiple levels of abstraction, it is then necessary to develop KIF

Agent: `mark-util`
Action: `(mark ?ass ?f)`
Description:
Run the Visual Basic marking utility to mark assignment `?ass`. The input file is `?f`.

Preconditions:
```
(file ?f (delimited #\, (listof string string string))) deleted
(file-represents-relation ?f STUDENT info202-dm
   (stuid stuname nw_id))                              deleted
```

Postconditions:
```
(file ?f (delimited #\, (listof string string string number)))
(file-represents-relation ?f
   (join STUDENT (project (select ASSESS (= cmptid ?ass))
                          (stuid mark)))
   info202-dm
   (stuid stuname nw_id mark))
```

Agent: `dbase-agent`
Action: `(add-marks ?f ?ass ?db)`
Description:
Add marks for assignment `?ass` in file `?f` to the database `?db`.

Preconditions:
```
(database-matches-datamodel ?db info202-dm)
(foreign-key-ok ?ass CMPT ?db)
(file ?f (delimited #\, (listof string string number)))
(file-represents-relation ?f
   (select ASSESS (= cmptid ?ass))
   info202-dm
   (stuid cmptid mark))
```

Postconditions:
```
(database-represents-relation ?db
   (select ASSESS (= cmptid ?ass))
   info202-dm)
```

Figure 5: Agent action specifications for steps 2 and 4 of the assignment marking process. Preconditions marked '*deleted*' no longer hold after the action is performed.

```
(file ?f (delimited #\, (list-of string string string number)))
(file-represents-relation ?f
   (join STUDENT (project (select ASSESS (= cmptid ?ass))
                          (stuid mark)))
   info202-dm
   (stuid stuname nw_id mark))
```

⇓ Delete columns 2 and 3 from file ?f

```
(file ?f (delimited #\, (list-of string number)))
(file-represents-relation ?f
   (project (select ASSESS (= cmptid ?ass))
            (stuid mark))
   info202-dm
   (stuid mark))
```

⇓ Insert new column 2 with value of ?ass in every row

```
(file ?f (delimited #\, (list-of string string number)))
(file-represents-relation ?f
   (select ASSESS (= cmptid ?ass))
   info202-dm
   (stuid cmptid mark))
```

Figure 6: The file manipulation process (step 3)

specifications that connect the representation of knowledge at the respective levels. This paper has discussed some of the issues associated with this kind of ABSI and the mapping across ontological levels. The authors are currently developing a desktop agent system for university course administration that will demonstrate the approach outlined in the paper.

References

[1] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[2] T. Gruber. Ontolingua: a mechanism to support portable ontologies. Technical Report KSL-91-66, Knowledge Systems Laboratory, Stanford University, 1991.

[3] Narinder Singh. A Common Lisp API and facilitator for ABSI: version 2.0.3. Technical Report Logic-93-4, Logic Group, Computer Science Department, Stanford University, 1993.

[4] M. R. Cutkosky, R. S. Engelmore, R. E. Fikes, M. R. Genesereth, and T. R. Gruber. PACT: An experiment in integrating engineering systems. *Computer*, 26(1):28–37, 1993.

[5] J. G. McGuire, D. R. Kuokka, J. C. Weber, J. M. Tenenbaum, T. R. Gruber, and G. R. Olsen. SHADE: Technology for knowledge-based collaborative engineering. *Concurrent Engineering: Research and Applications*, 1(3), 1993.

[6] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G. J. Sussman, J. Wisdom, and K. Yip. Intelligence in scientific computing. Technical Report AI Memo 1094, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1988.

[7] J. R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, 1969.

[8] T. Khedro and M. Genesereth. The federation architecture for interoperable agent-based concurrent engineering systems. *International Journal on Concurrent Engineering, Research and Applications*, 2:125–131, 1994.

[9] W. Wong and A. Keller. Developing an internet presence with online electronic catalogs. Stanford Center for Information Technology, http://www-db.stanford.edu/pub/keller/1994/cnet-online-cat.ps.

[10] Toyoaki Nishida and Hideaki Takeda. Towards the knowledgeable community. In *Proceedings of the International Conference on the Building and Sharing of Very Large Scale Knowledge Bases*, pages 157–166, 1993. http://ai-www.aist-nara.ac.jp/doc/people/takeda/doc/ps/kbks.ps.

[11] Sharable ontology library. Knowledge Systems Laboratory, Stanford University, http://www-ksl.stanford.edu/knowledge-sharing/ontologies/README.html.

[12] Narinder P. Singh and Mark A. Gisi. Coordinating distributed objects with declarative interfaces. Technical report, Computer Science Department, Stanford University, 1995. http://logic.stanford.edu/sharing/papers/oopsla.ps.

[13] C. J. Date. *An Introduction to Database Systems*. Addison Wesley, 6th edition, 1995.

[14] Omar A. Tawakol. Personal communication, August 1995.

[15] Amzi! WWW home page. http://www.amzi.com/.