TANDEM - a Design Method for Integrating Web Services into Multi-Agent Systems

Steve Goschnick | Sandrine Balbo
University of Melbourne
Interaction Design Group
+61 3 8344 1522 | 83441497
stevenbg|sandrine@unimelb.edu.au

Leon Sterling
University of Melbourne
Intelligent Agent Lab
111 Barry St, Carlton, 3010
leon@cs.mu.oz.au

Christine Sun
Independent Scholar
P.O. Box 218, Belgrave,
Australia, VIC, 3160
csun@taiwan.com.au

ABSTRACT
This paper introduces a new design method for multi-agent systems (MAS) that incorporate logic programming. The DigitalFriend is an example of a MAS with a built-in logic language interpreter, which is aimed at having the end-user as developer. It uses small fragments of logic as dynamic *glue*, bringing together numerous sub-agents that may exist as Web services. However, application developers need to devise well-formed predicates and logic rules - and there lies the need for a specific *design method*. The well-established techniques of Task Analysis and entity relation Normalization are drawn together into our new design method (TANDEM), introduced here with an example application - the *movie-cinema problem*.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent agents, Languages and structures, Multiagent systems.*

General Terms
Design, Languages, Human Factors.

Keywords
Multi-Agent System, Web Services Orchestration, End-user Systems, Logic Programming Methodology.

1. INTRODUCTION
This paper introduces a new *design method* suitable for developers of multi-agent systems that incorporate a *logic language,* such as the DigitalFriend [3], that wrap web services and other agents. The method draws upon *T*ask *A*nalysis [1] and data *N*ormalization from entity relation (ER) modeling [2], together into a new *DE*sign *M*ethod named *TANDEM*. - for devising the logic predicates and rules, which are distributed amongst the sub-agents. The method is demonstrated in the design of a *movie-cinema* locating application, typical of a modern application scenario that gathers constituent components from distributed web services. The method has broader applicability for logic programming in general.

AAMAS'06, May 8-12, 2006, Hakodate, Hokkaido, Japan.

2. THE *TANDEM* DESIGN METHOD
The analysis begins with a collection of possible *entities* uncovered by a requirements gathering exercise. In the working system, information representing these entities are often dynamically sourced from a number of unrelated distributed Web services. Nonetheless, ER modeling proceeds as if it were going to be implemented as a singular system.

2.1 The Movie-Cinema Problem
We present the methodology via an example application, one typical of projects developed by software engineering students at the University of Melbourne, centered on industry needs. Such projects often involve Internet resources and complex problems, which can benefit from recent research and development advances in software agents and MAS.

The problem: The client, a movie cinema operator, wants a system accessible by movie-going customers via the Internet, one which will allow users of smart hand-held devices, to search for specific movie details, ultimately leading to the booking of cinema tickets. Space prevents a full problem specification here of the *movie-cinema problem.*

2.2 Phase 1 - Applying Task Analysis
Task Analysis identifies a user's goals and the procedure or steps needed to achieve them. It includes the process of analyzing the structure of a task and decomposed the task into related sub-tasks. Some of the tasks identified for the *movie-cinema problem* were extracted using a task analysis and are listed below as *verb-first* directives:

Task 1 - *Movie-oriented search*:

- *Find* all recently released movies.
- *Find* all movies within a particular genre.
- *Find* which movies feature a specific actor.
- *Find* all movies by a specific director.

Task 2 - *Cinema-oriented search* (has a similarly sized list)

Task 3 - *Location-oriented search* (ditto)

Task 4 - *Seat-oriented search and ticket allocation* (ditto)

These four groups of tasks and sub-tasks - represent the first step in a *hierarchical task modeling* exercise [1], which is all we take from task analysis.

2.3 Phase 2 - Applying ER Normalization

In database applications a technique for designing new systems that is widely used and easy to teach, and is taught in both Information Systems and Computer Science, is entity relation (ER) modeling [2].

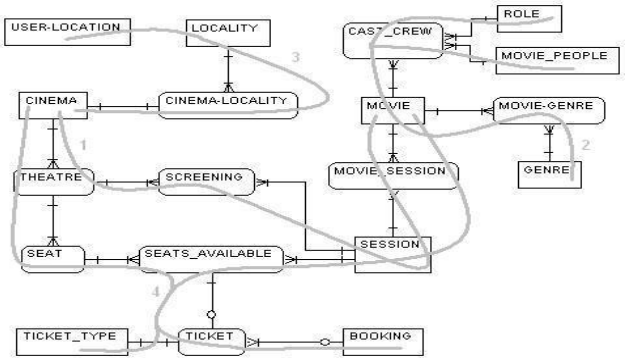

Fig.1 **– Normalized Entity Relation Diagram Modeling the Cinema Problem, with four *Logic Flow Lines***

And a technique that is applied to an ER model to get it into an *ideal form* - one that is most flexible with regard to queries put to it and to future change requirements - is called *Normalization*.

We adopt this well-known technique as a step within the TANDEM methodology - and since it is well-known, the process of normalization is not outlined here. We applied it to the movie-cinema problem above, and arrived at the normalized entity relation (ER) model in Figure 1 above.

Then, in order to divide the ER diagram into appropriate sub-areas, to be implemented as sub-agents (often powered by external web services), we identify *logic flow lines*, also depicted in Figure 1. These lines are mapped by taking the four high level tasks identified in the Task Analysis, and identifying which of the entities in the ER diagram are involved in each task - *without breaking the line*. – a condition that facilitates the process of *unification* [6] during the execution of the logic language.

2.4 Phase 3 - Create well-formed Logic Rules

In a logic program the predicate at the *head-of-the-rule* is a template for many of the queries that can be answered by the whole program. In the MAS application these rules and the associated logic is distributed between the various agents.

In our example solution five rules were derived, with the following five *head-of-the-rule* predicates, the first of which is explained in the next sub-section:

cinemaAssistant(CinemaName, Location, TheatreType, MovieTitle, GenreName, Rating, Date, StartTime)

movieAssistant(MovieTitle, Rating, GenreName, RunningTime, DvdRelease, RoleName, FirstName, Surname)

localityAssistant(Distance, Radius, Location, TownSuburb, Postcode, CinemaName)

seatSelection(CinemaName, Location, TheatreID, ImageMap, MovieTitle, Date, StartTime, RowNumber, SeatNumber, CoordX, CoordY, allocated)

ticketAssistant(CinemaID, TheatreID, Date, StartTime, RowNumber, SeatNumber, TicketNumber, TicketType, Cost, BookingID, FirstName, LastName, PaymentMethod, Paid)

A *query* is constructed by replacing one or more of the variables (i.e. those *terms* starting with a capital) with a specific literal instance, for example, the query in English:

What movies feature Sharon Stone as a star?

would be specified in a logic language (such as CoLoG within the DigitalFriend) as follows:

movieAssistant(***MovieTitle, Rating, GenreName, RunningTime, DvdRelease,*** *star, sharon, stone*)**?**

It can also be specified with under-scores in place of the variable names, leaving only the need to supply the literals:

movieAssistant(_ , _ , _ , _ , _ , *star, sharon, stone*)**?**

In the TANDEM methodology, we first decide on all the head-of-rule predicates that will be necessary to answer all of the tasks extracted via the Task Analysis. The four groupings of *primary tasks*, told us that we needed four predicates as outlined above (the fifth is via a branch in a logic flow line).

2.4.1 Deriving the Cinema Assistant Rule

To identify the actual *terms* needed by each predicate, we use a more detailed diagram than the ER diagram depicted in Figure 1. We could have chosen to depict all sub-area entity *attributes* in a more detailed style of ER diagram, however, in this paper, we have chosen to depict attributes for each of the sub-areas identified by a *logic flow line* in Figure 1, as UML Class diagrams – as we assumed that more agent application developers are likely to be familiar with the UML notation.

The constituent *terms* within each predicate are then determined by the atomic tasks that made up the high level task for which the predicate was instigated.

The next stage of this phase is to identify the best set of predicates on the *right-hand side* of each rule:

Head of the **Rule 1**:

cinemaAssistant(*CinemaName, Location, TheatreType, MovieTitle, GenreName, Rating, Date, StartTime*) ←

requires the following Predicates (from the normalized ER diagram) on the right hand side of the *implies* operator (←):

cinema(*CinemaID,* ***CinemaName, Location****, CinemaLatitude, CinemaLongitude*),

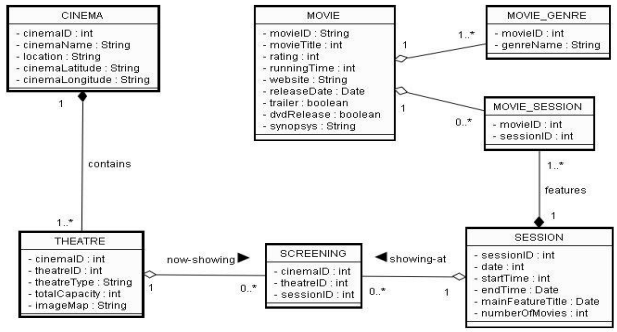

Fig.2 **– A UML Class diagram detailing attributes for entities associated with the *cinemaAssistant* rule.**

theatre(*CinemaID, TheatreID,* ***TheatreType****, TotalCapacity*),

session(*SessionID,* ***Date****,* ***StartTime****, EndTime, MainFeatureTitle, NumberOfMovies*),

movie(*MovieID,* ***MovieTitle****,* ***Rating****, RunningTime, Website, ReleaseDate, Trailer, DvdRelease, Synopsys*),

movieGenre(*MovieID,* ***GenreName***),

screening(*CinemaID, TheatreID, sessionID*),

movieSession(*MovieID, SessionID*);

Note: Commas between predicates are logical ANDs.

These *required* predicates making up the *body* of the rule, are identified via their member terms within the entity/class. i.e. Entities in Fig.2 with an *attribute* corresponding with a *term* in the head *predicate,* is included as a *rule body* predicate. Such attributes are highlighted in ***bold-italic*** font above.

3. IMPLEMENTING THE DESIGN

The DigitalFriend [3] - a user-friendly MAS - is the tool we used to *implement* the design. It is especially good at incorporating sub-agents that *wrap a web service* [4].

A tree view of the various agents is displayed via an octagonal-tile interface, recursively deep, but which displays three levels at a time. An insert in Figure 3 shows where *MovieAssistant*, *CinemaAssistant* and *LocalityAssistant* are grouped together under the EOC (envelope-of-capability) agent *MoviesCinemas.*

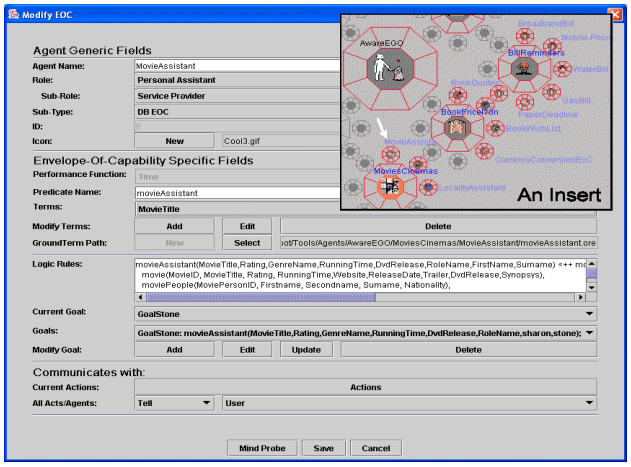

Fig.3 **– Modification of EoC Agent properties**

Both EOC agents and non-EOC sub-agents encompass predicates, which are created within the DigitalFriend interface via GUI dialog boxes. E.g. Figure 3 represents the property dialog box for an EOC agent – specifically for MovieAssistant.

There are four main sub-areas of input/edit fields in the dialog: *role and identity* fields; *predicate* and *term* related fields and buttons for interaction; logic *rules* and a series of predefined *goals/queries* for the agent; and an *inter-agent communication language* section (beyond the scope of this paper). The rule for *movieAssistant* – the second rule, as designed according to Phase 3 outlined in Section 2.4 – has been entered into the field *Logic Rules*. The goal/query that will return any movies involving the actress *Sharon Stone* as *star*, is the agent's *current goal*.

Sub-agents that retrieve records/ground-terms from web-services are similarly represented by a predicate with a list of terms – however, rather than rules, they also have *timers* set to retrieve new information at regular intervals set by the user, and they update stores of *ground-terms* that they maintain as agent *state*.

The DigitalFriend dynamically collects all rules, predicates and ground terms for an EOC agent and its sub-agents, and effectively combines them into a temporary logic program in the internal CoLoG language – built and executed when triggered [3]. Many of the resultant terms are passed to other agents – as specified via the inter-agent communication language in CoLoG – and many are passed as *messages* for the user's attention.

The CoLoG program which is dynamically constructed for a given EOC agent, can be inspected and executed (and tested and debugged) by the user from within the GUI, via the '*Mind Probe*' button at the bottom of the property dialog seen in Figure 3.

The only CoLoG code that the end-user had to devise is the *Rule*, entered via the 'Logic Rule' field in the property dialog in Figure 3. All other details have been entered via less textual GUI interaction, involving only buttons and single name terms.

4. DISCUSSION

Several current MAS development environments use internal logic languages, which then obligates the application developer to write well-formed logic predicates and rules. There is currently *a gap* in available methods for writing such predicates and rules [5].

ER modeling is more widely known than logic programming, and more teachable. Basic Task Analysis is straight forward. TANDEM combines these well-known techniques to find appropriate predicates, their constituent terms and the rules that bind them. It fills a gap in the methods for logic programming.

As seen via the *Movie-Cinema problem*, there is a need for such design methods in developing solutions that use such MAS. Compounding this need, the DigitalFriend is aimed at the *end-user*/developer market, where technical skills are at the spreadsheet-developer or the desktop-database-designer level. Our TANDEM design method provides *strong guidance* in finding appropriate predicates as early as possible, using well-tested techniques that can be learned in a short timeframe.

5. REFERENCES

[1] Annet, J. Hierarchical Task Analysis. In *The Handbook of Task Analysis for Human-Computer Interaction*, Diaper, D. and Stanton, N. (Eds), Lawrence Erlbaum, 2004, 67-82.

[2] Benyon, D. *Information and Data Modelling, Second Edition,* McGraw-Hill, England, 1997.

[3] Goschnick, S.B. & Graham, C. Augmenting Interaction and Cognition using Agent Architectures and Technology Inspired by Psychology and Social Worlds. To appear in: *Universal Access in the Information Society*, 4(3), 2005.

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

[5] Sterling, L. Patterns for Prolog Programming, In *Proceedings, Computational Logic: Logic Programming and Beyond 2002*: 374-401.

[6] Sterling, L. and Shapiro, E. *The Art of Prolog*, 2nd edition, The MIT Press, Massachusetts, 1994.