MELBOURNE UNIVERSITY 2008 SEMESTER 1

Implement Agent and Resource Allocation Algorithm in MIndiGolog

Project 2
433682 Software Agents & Agent programming languages

Xuqing Qi

5/2008

Abstract: This paper describes how to implement quality goal in MIndiGolog. A kitchen domain will be used and a dynamic location-based agent and resource allocation algorithm will be implemented. The quality goal is very simple, when system need agent or resource, all assign nearest agent or resource.

Introduction

Quality goal is a very import aspect of agent-oriented modelling. It is essential part of system design. [5]

MIndiGolog is a new approach to distributed problem solving based on high-level program execution [1].

MIndiGolog has follow features [4]:

- *Operators of ConGolog[2] (Concurrency, Interrupts)*
- *Controlled Search of IndiGolog[3]*
- *Richer Theory of Action:*
  - *Robust integration of true concurrency*
  - *Explicit temporal component*
  - *Seamless integration of natural actions*

This paper will show a possible solution to implement quality goal in MindiGolog.

Aim and Purpose

A kitchen domain will be used in this paper, which is based on my project 1 “Kitchen Model” [6]. This kitchen will be divided into 25 blocks and every kitchen hand, chef, utensil and kitchenware will be located into one block. (Figure 1)

block00 sink4	block01 table1	block02	block03	block04 sink2
block10 kitchenhands, chefs	block11	block12	block13	block14
block20 utensils	block21	block22	block23 oven1	block24 grill1
block30 raw material	block31	block32	block33	block34
block40 sink2	block41	block42	block43 table2	block44 sink1

Figure 1

Follow is the key constraints in this kitchen domain:

- The sink will be used to clean the raw material or utensil.
- The table will be used to perform "Cut" operation.
- Any raw material or utensil has to be cleaned before use.
- Agents can take utensil and raw material with them, but the table, sink, oven and grill is fixed in environment.
- Agent can go from one block to another block. The walking duration is determined the origination and destination. For example:
    - task_duration(kh01,goto(block00,block10))=1
    - task_duration(kh01,goto(block00,block04))=4

There is a quality goal for agent and resource allocation: only the nearest agent and resource should be allocated to the task.

For example, kh01 holds a bowl1 in table1 (block01) and want to clean bowl1. Sink1 and sink2 is available at that time. So when kitchen hand acquire sink, system should allocate sink2 to kitchen hand because sink2 is nearer than sink1.

Trace Location

The key feature of this kitchen domain is that system is able to trace the location of agents and resources. To implement this feature, we have to solve follow issue:

1. How to initial the location.
2. How to trace the agent location.
3. How to trace the resource location.

To initial the location, two procedures are introduced, initial_staff_location(Staff,B) and initial_obj_location(Obj,B). Everything should be initialled when the system starts.

To trace the agent location, one task are introduced, goto(Orig,Dest), Orig is the original block and Dest is the destine block. Then system can identify agent is in block or not by following logic:

```
staff_is_in_block(Staff,B,do(C,_,S)) :-
        %%Staff is initialled in block B
        (member(initial_staff_location(Staff,B),C))
        ;
        %%Staff just go into block B
        (
        member(end_task(Staff,goto(_,B)),C)
        )
        ;
        %%Staff is in block B before and not go out block B
        (
        staff_is_in_block(Staff,B,S),
        \+ member(begin_task(Staff,goto(B,_)),C)
        )
    .
```

Tracing resource location is more complicate than tracing agent location. First of all, kitchenware, tables and sinks are different from utensils and raw material. One is fixed in environment and another one can be taken from one block to another. Secondly, utensil and raw material doesn't go from block to block by itself. There has to be an agent taking utensil or raw material. System can trace utensil location by following logic:

```
obj_is_in_block(Obj,B,do(C,_,S)) :-
        (
                (obj_is_type(Obj,rawmaterial);obj_is_type(Obj,utensil)),
                (
                        %%Obj is initialled in block B
                        (member(initial_obj_location(Obj,B),C))
                        ;
                        %%Obj is taken by Staff and Staff go to block B
                        (
                        member(end_task(Staff,goto(_,B)),C),has_object(Staff,Obj,S)
                        )
                        ;
                        %%Obj is in block B before and no Staff takes it out that block
                        (
                        obj_is_in_block(Obj,B,S),
                        \+ (member(begin_task(Staff,goto(B,_)),C),has_object(Staff,Obj,S))
                        )
                )
        )
        ;
        (
                (obj_is_type(Obj,kitchen_ware);obj_is_type(Obj,table);obj_is_type(Obj,sink)),
                (
                        %%Obj is initialled in block B
                        member(initial_obj_location(Obj,B),C)
                        ;
                        %%Obj is in block B and always be there.
                        obj_is_in_block(Obj,B,S)
                )
        )
    .
```

Identify which one is the nearest one

How to identify an agent or an object is the nearest one? That's the most interesting part of the system. Before solving that problem, a new function has to be introduced to system:

is_near(A,B,Dist,do(_,_,_))

If block A and block B is nearer than integer Dist, it will return true. Otherwise, it will return false.

Then is_nearest function can be defined:

is_nearest_kh(Kh,Obj,do(C,T,S))=true

➔ Obj is the nearest object to Kh

➔ Not exist Obj1: the distance between Obj1 and Kh less than the distance between Obj and Kh

➔ Not exist Obj1, M: staff_is_in_block(Kh,B) and obj_is_in_block(Obj,B_Obj) and obj_is_in_block(Obj1,B_Obj1) and not is_near(B,B_Obj,M) and is_near(B,B_Obj1,M)

Base on that logic, the is_nearest_kh(Kh,Obj,do(C,T,S)) in MIndiGolog is:

```
is_nearest_kh(Kh,Obj,do(C,T,S)) :-
(       \+
        (
                kitchenhand(Kh),
                %% Obj and Obj1 is not equal to each other
                prim_obj(Obj),prim_obj(Obj1),Obj\=Obj1,
                %%Obj and Obj1 is avaliable
                \+ has_object(_,Obj,do(C,T,S)), \+ has_object(_,Obj1,do(C,T,S)),
                %% Obj and Obj1 is the same type, only sink, table and board
                %% will be used in this model
                (
                        (obj_is_type(Obj,sink),obj_is_type(Obj1,sink));
                        (obj_is_type(Obj,table),obj_is_type(Obj1,table));
                        (obj_is_type(Obj,board),obj_is_type(Obj1,board))
                ),
                %% B,B_Obj,B_Obj1 is the location of Kh, Obj and Obj1
                staff_is_in_block(Kh,B,do(C,T,S)),
                obj_is_in_block(Obj,B_Obj,do(C,T,S)),
                obj_is_in_block(Obj1,B_Obj1,do(C,T,S)),
                %% Max distance in this domain is 8
                (M=1;M=2;M=3;M=4;M=5;M=6;M=7;M=8),
                %% Obj1 is nearer to Kh than Obj
                (\+ is_near(B,B_Obj,M,do(C,T,S)),is_near(B,B_Obj1,M,do(C,T,S)))
        )
)
.
```

Similar function can be defined is_nearest_obj(Obj,Kh,do(C,T,S)), which means the Kh is the nearest kitchen hand to Obj.

These two is_nearest functions can be used to control agent or resource allocation.

The original code is:

proc(doClean(Kh,Obj),

pi(mySink, ?obj_is_type(mySink,sink)

…………………………………………

The proposed code will be:

```
proc(doClean(Kh,Obj),

        pi(mySink, ?and(obj_is_type(mySink,sink),is_nearest_kh(Kh,mySink))

        ………………………………………….
```

Performance Comparison

Follow table is the execution time comparison of make three different dishes: Salad, Beefsteak and Grilled Bacon.

	Salad	Beefsteak	Bacon
Original	150	75	122
Proposed	104	59	91

Table 1

The proposed program is more efficient than the original one.

Conclusion

What has been implemented in this paper is a possible solution to how to implement quality goal in MIndiGolog. Quality goal is a very important part of agent oriented modelling. MIndiGolog can handle quality goal with different kinds of constrains.

In this paper, constrains have been used when system “PI” agent or resource.

Another example is these quality goal related constrains can be put into poss() function, to check the quality goal before the task_begin or task_end.

Quality goal related constrains can be put into conflict function, to make sure the whole system match some quality goals.

Once the quality goal can be expressed as constrains, then the MIndiGolog can implement it.

Reference

[1]. Ryan F. Kelly and Adrian R. Pearce., Towards high level programming for distributed problem solving. *International Conference on Intelligent Agent Technology (IAT-06)*, Hong Kong, pages 490--497 (2006)

[2]. Giuseppe De Giacomo, Yves Lespérance, Hector J. Levesque, ConGolog, a concurrent programming language based on the situation calculus, Sep. 1999, Elsevier

[3]. Yves Lespérance and Hector J. Levesque, IndiGolog: A Programming Language for Cognitive Agent, http://www.ercim.org/publication/Ercim_News/enw53/lesperance.html

[4]. Ryan Kelly, MIndiGolog: Multi-Agent Golog, http://www.cs.mu.oz.au/482/lectures/mindigolog.pdf

[5]. Leon Sterling and Kuldar Taveter, The Art of Agent-Oriented Modelling, Semester 1, 2008

[6]. Xuqing Qi, Kitchen Model, Project 1 of 433682, Semester 1, 2008