EXAM

for the Lecture Course on

IMPACT

given from October 2–6 at CACIC 2000

Jürgen Dix

Saturday, October 7, 2000

Abstract

This exam is closed-book and closed-notes. *Clarity and neatness count.* There are 21 questions that can be briefly answered. **Good luck!**

1 QUESTIONS

1. General

1. What is the difference between general distributed AI and Multiagent systems?
2. Elaborate on the statement *Agents are pretty much like objects in object oriented frameworks!*
3. What kind of servers are there in *IMPACT*?
4. In the service description language, we are using *verb : noun(noun)* expressions to denote service names. How can we define a distance between service-names, given distances on the set of verbs and the set of noun(noun)'s?

2. The Code Call Mechanism

1. How do we abstract from given software in *IMPACT*?
2. What is the state of an agent in *IMPACT*? Does the mailbox belong do it? Why?
3. Which of the following code call conditions are safe:
   (a) $\mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$ & $\mathbf{in}(\mathtt{Z}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$.
   (b) $\mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$ & $\mathbf{in}(\mathtt{Z}, \mathcal{S} : g(\mathtt{c}, \mathtt{d}))$ & $\mathbf{in}(\mathtt{Y}, \mathcal{S} : g(\mathtt{e}, \mathtt{Z}))$.
   (c) $\mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$ & $\mathtt{Y} < \mathtt{Z}$ & $\mathbf{in}(\mathtt{Z}, \mathcal{S} : g(\mathtt{a}, \mathtt{b}))$.
4. Is the following true? Explain your answer.
   Any code call condition is safe with respect to the set of all variables occuring in it.

3. Actions and Agent Programs

1. How is an action described in *IMPACT*? Why is the precondition required to be safe?
2. Describe in your own words the notion of executability of an action $\alpha(\vec{X})$. The formal definition uses a notion of (θ,γ)-executability. What does the γ represent?
3. Describe one of the three different notions of concurrency provided in *IMPACT*.
4. What is the difference between action and integrity constraints?
5. Sketch the agent-decision-cycle of *IMPACT* agents.

4. Regular Agents

1. Why is the property of safeness not sufficient for determining the truth of code call conditions?
2. Illustrate the idea of a binding pattern in order to define strong safeness. Do we really need input of the designer of the agent?
3. What is the idea behind defining conflicting modalities?
4. Consider the following finiteness table for the code calls in 4. of Section 2 on the previous page:

Code Call	Binding Pattern
$\mathcal{S}:f(\mathtt{X},\mathtt{Y},\mathtt{Z})$	$(a,\flat,c)$
$\mathcal{S}:g(\mathtt{X},\mathtt{Y})$	$(\flat,d)$
$\mathcal{S}:g(\mathtt{X},\mathtt{Y})$	$(e,\flat)$

   Which of (a), (b) and (c) are strongly safe?
5. What is the reason that the notion of a program *being conflict-free* is undecidable (as opposed to the notion of being safe)?

5. Extensions

1. Belief programs can be reduced to ordinary programs. Which additional datastructures are needed to do this?
2. How does the original notion of a code call change in probabilistic agent programs?
3. What is the idea behind temporal annotations in temporal programs? How do these temporal annotations differ from annotations in probabilistic agent programs?

2 SOLUTIONS

1. General

1. While both MAS and DAI consider distributed entities, in DAI the solution method for each entity is fixed upfront. In contrast to this, in MAS each agent is free to chose its own solution: only the protocol is fixed.

2. Agents have their own control thread, whereas in OO there is usually only one control thread for the whole system. Also objects with public methods have no control on who is using these methods, unlike agents.

3. There are *Registration–*, *Yellow Pages–*, *Thesauri–*, and *Type–* servers.

4. We can define a composite distance function by adding the distances of the verbs and the noun-terms. E.g. the distance between $v_1 : n_1(n_1')$ and $v_2 : n_2(n_2')$ is the sum of the distance of v_1 and v_2 and the distance between $n_1(n_1')$ and $n_1(n_1')$.

2. The Code Call Mechanism

1. We view software as a triple consisting of a set of datastructures, a set of functions operating on them and a set of composition operators to build new datastructures.

2. (a) $\mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$ & $\mathbf{in}(\mathtt{Z}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$.
   This code call condition is not safe because the variable Y is never instantiated and therefore $\mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c})$ can not be executed.

   (b) $\mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$ & $\mathbf{in}(\mathtt{Z}, \mathcal{S} : g(\mathtt{c}, \mathtt{d}))$ & $\mathbf{in}(\mathtt{Y}, \mathcal{S} : g(\mathtt{e}, \mathtt{Z}))$.
   This code call condition is safe. When written as

   $$\mathbf{in}(\mathtt{Z}, \mathcal{S} : g(\mathtt{c}, \mathtt{d})) \ \& \ \mathbf{in}(\mathtt{Y}, \mathcal{S} : g(\mathtt{e}, \mathtt{Z})) \ \& \ \mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$$

   it can be executed from left to right.

   (c) $\mathbf{in}(\mathtt{X}, \mathcal{S} : f(\mathtt{a}, \mathtt{Y}, \mathtt{c}))$ & $\mathtt{Y} < \mathtt{Z}$ & $\mathbf{in}(\mathtt{Z}, \mathcal{S} : g(\mathtt{a}, \mathtt{b}))$.
   This code call condition is safe. It can be executed from right to left.

3. The statement is true. When all variables are ground, each code call condition can be evaluated.

3. Extensions

1. An action has, besides a name and a schema, three important components: a precondition (to determine whether it can be applied), an add list (to ensure that certain ccc's are true in the state after executing the action), and a delete list (to ensure certain ccc's are no more true in the state after executing the action).

2. An action can be only executable when all its arguments are ground. If this is the case, we have to check whether the precondition is satisfied in the current state. If it is the action can be executed. Then we have to make all ccc's in the add list to hold in the state. Also all cc's in the delete list should no more hold in the state. This has to be done for all instantiations.

   The γ stands for an instantiation of all additional variables in the precondition. These can be

3. The weakest notion provided is when we simply take the union of all the add lists as well as the union of all the delete lists as the merged action.

   A more refined notion is to find a particular ordering of actions that do not conflict and then execute the actions in this order.

   The most advanced notion is when all possible orderings are checked, they do not conflict and they all lead to the same final state.

4. Integrity constraints enure that the state preserves certain properties. Action constraints ensure that certain actions are not concurrently executed.

5. (1) Messages are evaluted. (2) According to the chosen semantics a status set is computed. (3) According to the concurrency notion certain actions are executed.

4. Regular Agents

1. It is not sufficient because a code call can return an infinite set. Thus evaluating a code call condition consisting of two code calls may never terminate because the second component is never reached.

2. The agent designer has to state conditions under which his functions return finite sets. He can do so by specifying certain arguments and listen them in a finiteness table. Sometimes, it might suffice to specify only some arguments, all others can be arbitrary. This is the information stored in a binding pattern.

   Input of the designer is needed, because the general problem is undecidable.

3. The idea is to ensure that agent programs are conflict-free. Therefore the heads of rules need not to be in conflict. But these heads start with deontic modalities. Some of these modalities are in conflict, others are not. For example **P** is in conflict with **F**.

4. Only (b) is strongly safe.

5. The reason is that this notion is defined with respect to arbitrary states. And states are rich enough to express any knowledge. In particular, the halting problem for Turing machines can be encoded. This is not the case for the purely syntactical notion of safeness.

5. Extensions

1. We need a belief table and a belief semantics table.

2. Instead of a code call returnng objects, we are considering code calls returning arbitrary random variables. These random variables consist of a set of objects together with a probability distribution over them.

3. Temporal annotations represent time intervals. As opposed to this, probabilistic annotations represent probability intervals.