The 5Rings “Team Report”

Paulo T. Silva[1], Helder Coelho[2]

[1] Instituto Superior de Engenharia de Lisboa, Departamento da Engenharia da Electrónica e Telecom. e de Computadores, R. Conselheiro Emídio Navarro, 1, 1949-014 Lisboa, Portugal
[2] Faculdade de Ciências da Universidade de Lisboa, Departamento de Informática, Bloco C5, Piso 1, Campo Grande, 1749-016 Lisboa, Portugal
[1]ptrigo@isel.ipl.pt, [2]hcoelho@di.fc.ul.pt

1 Introduction

This technical report describes the essential concepts of the 5Rings team [1] agents and illustrates the most relevant implementation issues.

The section 2 presents a formal frame of the agent concept. At section 3, the formal frame is used to specify the *RoboCupRescue* agents. The section 4 describes the implementation of each agent constituent (from the formal frame). The last section outlines the current main strength and weakness of our implementation.

2 The tuple representation of an agent

At this section an attempt is made to formally frame the agent concept. We pretend to build a systematic and higher view of our agents’ implementation.

The following 7-tuple defines the structure of an agent *ag* immersed in a multi-agent system:

$ag = < G, M, Wm(E_{self}, E_{other-}, Mr), P_{Ac}, H_{Ac},, E, C >$

where,

- G is a set of goals that the agent aims to achieve as a whole,
- M is the theoretic model implemented by the agent,
- Wm is an working memory inhabited by representations – self and other agents’ environment representation (E_{self} and E_{other} respectively) and any representations built to support the agent model (designated by the Mr symbol),
- P_{Ac} is a set of primary actions or primary protocols that the agent can use,
- H_{Ac} is a set of higher level actions or protocols that the agent can use,
- E is an environment where the agent act,
- C represents the constraints on the channels used for inter-agent communication and for the agent’s interaction with E.

The agent has three main sources of G goals: i) the agent designer, that usually pre-defines goals, ii) the other agents, with whom an hierarchical relationship may exist, and iii) the human user, when such a direct interaction exists.

The agent behavior implementation is superimposed by theoretical model M guidelines. The model ranges from a pure reactive and socially deaf to a deliberative and socially fulfiller agent.

The agent working memory contains its own environment representation (e.g. its current state, historic evolution, aggregated knowledge) and the agent's view of others' agents' perspective on the environment. All representations that need to exist in order to support the agent model M are also contained at Wm.

The P_{Ac} symbol represents the primary (basic) capabilities used by the agent to communicate with others or to interact with the surrounding environment.

The H_{Ac} symbol represents the higher level capabilities designed over P_{Ac}.

The environment E, evolves accounting for agent actions and to models of change based on physical laws or artificial rules.

The C links are used by the agents to communicate. We explicitly distinguish two C subsets: the *ag* to *ag'* (inter-agent) communication subset and the *ag* to E (agent to environment) communication subset. From here we state that $C = C_{\text{ag}} \cup C_{\text{agE}}$, where C_{ag} represents the communication channels among agents and C_{agE}, represents the interaction channels between agents and E. The $C_{\text{ag}} = \emptyset$ means that agents act with no communication (among them). The $C_{\text{agE}} = \emptyset$ means that agents can not interact with the environment and thus the simulation is not taking place (perhaps a replay of a previous simulation).

3 Rescue agents based on the tuple representation of an agent

The *RoboCupRescue* multi-agent system is inhabited by a set of agent types. Each agent type represents a set of agents with a specific capabilities' pattern. Thus, there are differentiation capabilities, which only apply to a particular agent type (e.g. "extinguish" only apply to Fire Brigade agent type), and common capabilities that apply to all agent types (vision, hearing, speech and locomotion).

The number of agents that belong to each agent type depends on the *RoboCupRescue* competition rules.

The set *AgType* of agent types is:

AgType = { Civilian, Fire Brigade, Ambulance Team, Police Force, Fire Station, Ambulance Center, Police Office }

Each $ag \in AgType$ is represented by its own tuple. To get things simple and clear we will first describe the basic "Fire Brigade" agent.

The basic “Fire Brigade” agent follows Morimoto’s agent description [2] and was adopted by the 5Rings team as the first step towards more sophisticated rescue agents.
The basic “Fire Brigade” tuple representation is the following:

- G = { extinguish fires, inform on endangered agents },
- M is the very general “action is an immediate response to an external event” programming model,
- Wm: $E_{self} \equiv E_{other}$ is the map representation of the disaster space; Mr contains all state variables that support M,
- P_{Ac} = { move(node_array), extinguish(nozzle_array), say, tell, hear say, hear tell },
- H_{Ac} = { route finder, fire selector },
- E is a city region (at *Kobe*, *Foligno* or random) for the five hours that take place after an earthquake,
- $C = C_{\mathrm{ag}} \cup C_{\mathrm{agE}}$, is physically materialized by one single channel; the usage of that same channel is sampled in time, so we may think of two logical channels with following constraint between them: i) C_{ag} is only available between two consecutive instants where C_{agE} is available, and ii) the time period between two consecutive C_{agE} availability is about 500 milliseconds.

For each agent type, the constraints on C also depend on the P_{Ac} element. We represent such constraint as a 2-tuple where the first element is a P_{Ac} subset and the second element is a numeric value representing the upper bound capacity of the channel. The C constraints are the following:

C_{ag} = { < {say, tell}, 4 >, < {hear say, hear tell}, 4 > }, meaning that, there is an upper bound of 4 outgoing (say, tell) and 4 incoming (hear say, hear tell) messages between two consecutive uses of C_{agE}.

C_{agE} = { < {move(node_array), extinguish(nozzle_array)}, 1 >, meaning that, there is an upper bound of 1 message to post an action intention towards E.

All other $ag \in AgType$ are expressible as simple variations of G, P_{Ac}, H_{Ac} and C elements. More sophisticated agents require sophistication of M, Wm and H_{Ac} elements.

4 The implementation of the tuple representation of an agent

The tuple representation of an agent was implemented for the *RoboCupRescue 2004* competition and variations on G, P_{Ac}, H_{Ac} and C were sufficient to construct all the different rescue agents’ structures.
The builder design pattern [3] was used to implement a flexible *ag* tuple representation. The Table 1 presents a code excerpt.

```
concreteBuilder.buildConnector( initiatorCommunicator );   // C
concreteBuilder.buildAgentToPlatformConnection();          // C
concreteBuilder.buildAgentGoalSet();                       // G
concreteBuilder.buildAgentModel();                         // M
concreteBuilder.buildAgentWorkingMemory();                 // Wm
concreteBuilder.buildAgentPrimaryCapability();             // PAc
concreteBuilder.buildAgent();                              // ag
```

Table 1. The tuple implementation.

Two different *M* elements (agent models) were implemented:

- the first was called “baseline” and was used at the qualification stage; it followed the very general “action is an immediate response to an external event” programming model,
- the second was called “RRR” and was used at the final competition stage; the model was built around the theoretic guidelines of reactive, deliberative and socially oriented agents; the “RRR” model proposes an equilibrium among the reactive level (R0), the self motivation reasoning level (R1) and the social strength towards global achievements (R2). The “RRR” model architectural guideline is graphically represented at the Fig. 1.

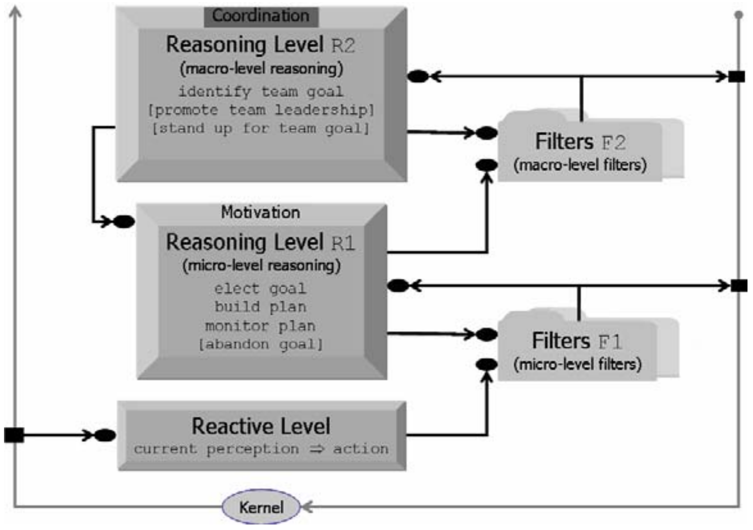

Fig. 1. The “RRR” agent architecture.

To implement the reactive level, a simplified rule based engine was developed. At that level, rules such as self preservation are specified to all agents (e.g. if my health point is negative then move to a refuge; if I am buried then cry for help) as well as some rules about agent’s specific but basic behavior (e.g. for an Ambulance Team, if I am at the location of a buried agent and the estimated time of its rescue is lower then of its death, then rescue that agent).

The motivation reasoning level (R1) is implemented as a belief, desire and intention cycle, with an intention persistence evaluation. The intention persistence guarantees fidelity to the current goal during a predefined number of plan repairs. A plan repair occurs when the plan cost deviation (difference between the estimated and the measured cost) exceeds a certain threshold. The plan is monitored at each simulation step and deviation updated.

The RRR architecture was mapped to *M* as a control sequence of the agent behavior at each simulation cycle. The Table 2 presents an excerpt of the *M* realization.

```
r0.ruleEngine.updateAgenda( r0.ruleSet );
r0.ruleEngine.runAgendaAllInteractionRules();

if( ! r0.ruleEngine.isAgendaActionSetEmpty() )
{
  E_Rule firstRule = r0.ruleEngine.getAgendaFirstAction();
  String firstActionName = firstRule.getStatementName();
  boolean r0_ACCEPT = r0.filter( firstActionName, firstRule );
  if( r0.ACCEPT )
  {
    r1.inform( firstActionName, firstRule );
    r0.ruleEngine.runAgendaFirstAction();
  }
}

r1.updateBelief();
r1.communicateWithOtherAgents();

ArrayList newPlan = r1.buildNewPlan( r1.buildDesire() );

r1.persistPreviousIntention( newPlan );
r1.acceptNewIntention( newPlan );
r1.gotoContingencyPlan();
```

Table 2. The RRR model (*M*) realization.

The coordination reasoning level (R2) is not explicitly implemented at the above algorithm. The implemented coordination is based on a basic convention that mainly distributes the agent’s responsibilities over different geographical sections of the disaster space. These sections are predefined and known by all agents (the same responsibility allocation algorithm is executed by all agents).

4.1 The reactive level (R0)

The reactive level contains a set of rules with the general “condition implies behavior” format. Two rule subsets are defined: i) one for interaction, where “condition implies communication”, and ii) other for the action rules, where “condition implies action”. A total order is imposed over rules of each subset.

The Table 3 shows each rule and the interaction (I) and action (A) lists (ordered subsets) for each agent type: Ambulance Team (AT), Police Force (PF), and Fire Brigade (FB). Each number indicates the position of the rule at its own list (I or A); conditions have qualifiers (<self>, <other> and <here>) that apply to the statement following the qualifier; behaviors have outcome indicators ([path]) to evidence the necessity of a planner or decision maker outcome.

AT		PF		FB			
I	A	I	A	I	A	*condition*	*behavior*
	5		2		2	<self> alive & not buried & damage & not refuge	move(refuge) [path]
1		1		1		*<self> alive & buried*	*tell*
	3		1		1	<self> damage & refuge	rest
	-		3		-	<self> road_blockade	clear
	1		-			<self> loaded & not refuge	move(refuge) [path]
	2		-			<self> loaded & refuge	unload
	4		-			<self> not loaded & not refuge & <other> <here> alive & damage & not buried	load(max damage)
	6		-			<self> not loaded <other> <here> alive buried	rescue(max damage)
-		2		2		*<other> <here> alive & buried*	*tell*
-		3		3		*<other> <here> alive & buried*	*say*

Table 3. The "condition implies behaviour" rules and each rule's order, within interaction (I) and action (A) lists, for each agent type (AT, PF and FB).

Each rule list (I and A) is evaluated at each simulation cycle. All interaction rules (I) satisfying its conditions are fired; the first action rule (A) satisfying its condition become ready to fire (but does not fire).

When a rule fires, its behavior is sent to the *RoboCupRescue* kernel process. When a rule becomes ready to fire, it goes through the R0 filter. If the filter accepts the rule, then the rule is fired. Otherwise the action proposed by the ready to fire rule is confronted with the outcome of the R1 reasoning.

4.2 The micro-level reasoning (R1)

This reasoning level starts with a belief update function. At each simulation step the function updates the beliefs concerning the buildings already visited by the agent (damaged civilian's exploration) and the roads already visited by the agent (blockade exploration).

The desires are built after the belief update activity. Each desire is a list of the most interesting (from the agent's perspective) arguments for the possible agent actions. A total order is defined over desires.

Thus for the Ambulance Team (AT) agent type, the following desires, for the move action, are defined: i) any humanoids buried within the agent's vision radius, ii) any other AT agents at known locations, iii) any other rescue teams (Police Force or Fire Brigade) at known locations, iv) any civilians at known locations, v) any civilians at unknown locations (local exploration), and vi) any not burning building within the agent's geographical section.

For Police Force (PF) agent type the following desires, for the move action, are defined: i) any other rescue team reported blockade, ii) blockades within the agent's vision radius, and iii) any unvisited road within the agent's geographical section.

Although the Fire Brigade (FB) implementation of the "RRR" model was not fully accomplished, the FB desires are: i) extinguish an early fire within its own geographical section, ii) extinguish an early fire elsewhere, and iii) extinguish any fire anywhere.

The desires predefined order is used to select the first (following desire's order) non empty list of preferable desires (most interesting action arguments).

The list of preferable desires is used to build a plan to achieve one of the elements in that list. A plan is a sequence of actions that takes the agent from its current state to a state described by any element contained at the preferable desires list. The planner minimizes a cost function, so the minimal cost desire is found during the construction of the plan.

The minimal cost desire becomes the agent's intention and a plan monitor is used to account for delays and repairs of the original plan. The intention persists until a predefined number of plan repairs occur; after that a new intention is allowed to be elected.

A path planner and path plan execution monitor were implemented. The path planner implemented cost functions are: i) passable roads, where lower costs are assignment to roads the agent knows to be free of blockades, ii) shorter roads, where lower costs are assigned to lower length roads, and iii) wider shorter roads, where lower costs are assigned to wider (with more lanes) roads assuming the same length (uses length/lanes ratio).

The path planner has two additional capabilities: i) the "must go" list, and ii) the "taboo" list. The "must go" list is used to indicate that the agent must go through, at least one of the locations contained in the list, before reaching its final destination. The "taboo" list gives the reverse indication, that is, the agent must avoid all the "taboo" list locations.

The general path planner algorithm composes two sub-problems. The first is to get a path from the current location to a "must go" location. The second is to get a path, from the solution of the previous sub-problem, to one of the destination locations. The Table 4 shows the general planner algorithm.

```
public ArrayList getWiderShorter(
   Collection destination, Collection mustGo, Collection taboo )
{
  if( mustGo.isEmpty() )
  { return getWiderShorter( destination, taboo ); }

  // The 1st SubProblem (from current to a mustGo location)
  GIS_RoutingProblem firstSubProblem =
   new GIS_RoutingProblem_widerShorter(
          self.motionlessPosition(), mustGo, taboo );

  // The 2nd SubProblem (from "mustGo" to a destination location)
  // Note: initial state of 2nd subproblem is still unknown
  // (it will be the solution state obtained from 1st subproblem)
  B_GIS_RoutingProblem secondSubProblem =
   new B_GIS_RoutingProblem_widerShorter(
          destination, taboo );

  // Return the composition of 1st and 2nd SubProblems
  return getSolutionFromTwoSubProblemComposition(
      firstSubProblem, secondSubProblem );
}
```

Table 4. The general planner algorithm.

The plan is monitored at each simulation cycle. Both the agent's expected location at the next simulation cycle and the accumulated deviation (from expectations) are calculated. The Table 5 shows an excerpt of the method invoked at each plan monitoring time.

```
public void updateTime( int time )
{
...
  int elapsedTime = time – previousTime;
  int elapsedTimeToConsider = Math.max(elapsedTime – unexpectedCost, 0);
  unexpectedCost = Math.max(unexpectedCost - elapsedTime, 0);

  int maximumExpectedProgress =
      elapsedTimeToConsider * maximumExpectedProgressAtEachTime;

  int effectiveProgress = distanceFromTo(
      indexOfPreviousLocation, indexOfCurrentLocation, planToMonitor  );

  int deviation = maximumExpectedProgress - effectiveProgress;
  accumulatedDeviation = accumulatedDeviation + deviation;
...
}
```

Table 5. Accumulated deviation update during plan monitoring.

When the accumulated deviation exceeds a certain threshold a plan repair is proposed. Then, after a predefined number of plan repairs a new intention may be elected. The Table 6 shows an excerpt of the method invoked during the evaluation of intention persistence; the intention persists for a predefined number of plan repairs.

```
public void persistPreviousIntention( ArrayList aarr_newPlan )
{
...
 PlanMonitor monitor = getMonitor();
 if( monitor.isExecuting() )
  if( monitor.getTotalNumberOfPlanRepair() < persistence_move )
   moveWithMonitorToDestination( monitor.getPlanDestination() );
...
}
```

Table 6. Intention persistence evaluation.

4.3 The macro-level reasoning (R2)

The coordination reasoning level (R2) assumes a basic convention that mainly distributes the agent's responsibilities over different geographical sections of the disaster space.

The implemented geographical responsibility defines a grid partition over the disaster space. The Fig. 2 show a disaster space (Kobe) and the bottom left element (section) of its grid partition.

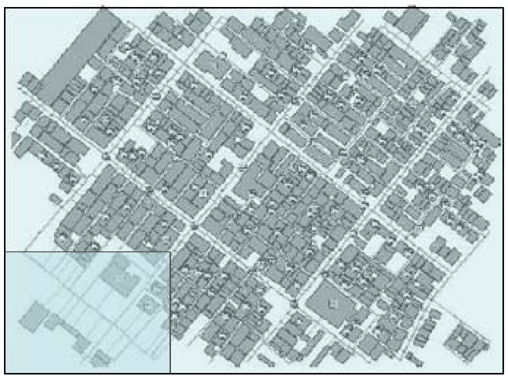
Fig. 2. Kobe disaster space and the bottom left element of its grid partition.

The grid number of rows and columns depends on the type of agent and on the number of agents of that type (e.g. if there are 15 Police Force agents a 4x3 grid may be defined).

Agents are distributed so that all grid elements (sections) have the same number of agents and that at least one agent is free (not allocated to a specific section). These free agents are allocated to a 1x1 grid partition (a section that contains the whole disaster space).

The grid allocation process occurs without inter-agent communication. The order relation between agents is mapped in an order relation between sections. Thus each agent decides by itself to get the section that corresponds to its "age" (the position in the array that contains all agent identities). Agents younger (higher index array) than the grid dimension get the whole disaster space.

At each section there is also a home position – the node closer to the section's mid point. A maximum distance (from mid point) is defined and if no node exists within that distance, than the agent gets the full disaster space. This tries to avoid allocating agents to sections with nodes mainly concentrated at the section borders. The Table 7 shows the essential code for the section allocation process.

```
// Agent class method
protected E_SectionWithRoads getMySection()
{ return mySpaceGrid.getHOME( getAge() ); }

// SpaceGrid class method (a WorkingMemory component)
public E_SectionWithRoads getHOME( int agentAge )
{ E_SectionWithRoads HOME = getSectionByID( agentAge );
  if( HOME.getHOME() == null )
  { return fullSpaceSection; }
  return HOME; }
```

Table 7. Allocation of sections (each grid element).

The importance of this convention depends on the order of the desire that accounts for the convention. Thus at the current implementation, the reasoning concerning this convention really occurs at R1 level and not at R2.

An additional convention is defined as a “fast crossing between section borders”. For each section, the node closest to each border, is detected and the “wider shorter” path is calculated as outlined in the Fig. 3.

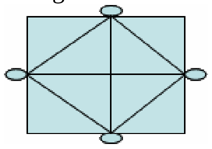

Fig. 3. The fast crossing roads between section borders.

According to this convention Police Force agents must primarily ensure that those roads are clear; all other agents that pretend to use “fast and safe” roads should rely on this convention.

5 The agent’s rescue performance

At this point it is important to remark that the agent’s tuple structural definition proved to be flexible, concerning the exchange of the agent models and the implementation of the capabilities of the different agent types. The Police Force and the Ambulance Teams implementation both follow the RRR model; the Fire Brigade is the more incomplete implementation of RRR. The command centers (Police Office, Ambulance Center and Fire Station) implementation is still very immature; they mainly act at communication relay agents.

The agent’s tuple structural definition alone does not imply a “good” rescue performance. From this report it is clear that the weakness of our agents follows from some main issues that remain to be handled: i) the explicit specification of the team work concept within the agent model, ii) the leader concept as a means to deal with coordination and communication scarceness, iii) a sophisticated situation evaluation and decision making technique, and iv) an effective usage of command centers (institutional leadership).

6 References

[1] Silva, P. Araújo, P. Remédios, A. Lopes, C. Basílio, B. Loureiro, T. Moniz, L. and Coelho, H. (2004). “The 5Rings Team Description Paper". Proc of the RoboCup2004 Symposium. July 4-5, Lisbon, Portugal.

[2] Morimoto, T. (2002). How to Develop a RoboCupRescue Agent for RoboCupRescue Simulation System. version 0, first edition. RoboCupRescue Technical Committee.

[3] Gamma E., Helm R., Johnson R., and Vlissides J. (1995). “Design Patterns – Elements of Reusable Object-Oriented Software”. Addison-Wesley.