A FRAMEWORK FOR COMPLEX SKILL GENERATION

D.M. Rivero\textsuperscript{2}, R. Barber\textsuperscript{1}, F.J. Rodriguez\textsuperscript{1}, M.A. Salichs\textsuperscript{1}

\textsuperscript{1} Carlos III University, Madrid, Spain
\textsuperscript{2} The Andes University, Los Andes, Venezuela
rivero@ing.uc3m.es, rbarber@ing.uc3m.es, urbano@ing.uc3m.es, salichs@ing.uc3m.es

Abstract: The software developments used in the hybrid architectures in which reaction and deliberation are combined and coordinated have accomplished, at the present time more efficient architectures, though the solutions provided are ad-hoc. Each solution is a unique development, which make difficult the cooperation process between labs. At the present time, there exits a great interest in the development of software architectures or of generic components, with open code and which work in a different platforms, in a simple way. This paper presents the design of a framework and its use, in order to develop complex skill in a flexible and simple way. It is based on a set of design patterns which make easier its use and documentation. Each skill is treated as a distributed object that is used as answer for other skills request, designing each one of them as a distributed object in which the skills belonging to lower levels can be combined to obtain more complex behaviors. These skills combination as the AD architecture states, in most cases can be obtained by skill sequencing by means of a skill sequencer. Copyright © 2004 IFAC.

Keywords: Mobile Robots, Software Architecture, Skill Sequencer, Software Framework.

1. INTRODUCTION

As the mobile robots software architectures have evolved, satisfying properties such as: flexibility, scalability, robustness, etc., the interest of using them, in different commercial applications, has increased. Different software architectures have been developed. Some, like BERRA (Lind, 2000), Saphira (Kono, 1996), MIRO (Ende, 2002), OPEN_R (Fuji, 1997), have used as a robot design strategy a hybrid architecture. Even though, these software development do not make easier the algorithm transfer or the interoperability of developments between labs. Among the reasons which make this cooperation difficult the following can be mentioned: the interfaces differences, the programming paradigm and the use and the encapsulation of data.

At the present time, the goal of commercial developments such as: Mobility, OPEN_R is to provide users with a set of components which serve as support for developments in robots. OROCOS (Bruy, 2002) is a project currently carried out by three partners: LASS in Toulouse, France, KHT in Stockholm, Sweden, and KU Leuven in Belgium. This project’s main goal is to define a standard for the robots architecture’s components of robots. Having in mind the recent software works in robots architectures (Oreback, 2003) (Brugaly, 2002) the characteristics that a standard software development must satisfy, can be established. Some of these characteristics are: common interfaces for accessing data of the actuator and sensors, mechanisms that
standardized the access and the behaviors enabling, object orientated languages to make easy the suitability, flexibility, scalability and reusability of the developments based on hybrid architectures.

In this paper, the design of a framework developed to build complex and deliberative skills is presented. This design was based on the experience obtained working with Mobility and in the characteristics, described above, on standard developments. The software design uses as robot architecture the hybrid architecture proposed by R. Barber (Barber, 2000) called AD (Automatic/Deliberative) architecture. This architecture is divided in two levels: a higher level, deliberative type, and a lower level, automatic type.

The three layer architecture has an intermediate layer which controls and sequences the behaviors such as: AURA (Arkin, 1986), Bonasso’s 3T architecture (Bonasso, 1997) with a middle layer named Sequencing Layer, BERRA which names this layer Task Execution or Alami et al. (Alami, 1997) which calls this layer Execution Control Level. Meanwhile, the AD architecture defines the skills sequencing as part of those skills, that independently of the level which they belong to, need to sequence behaviors. As an example, the following skills can be listed: navigation skill of the Deliberative Level that manage automatic skills like go to corridor or the complex skill in the Automatic Level, that manages other automatic skills too.

This paper’s skeleton is made up of section 2, where the general characteristics of the sequencer agent are described. In section 3 the component design to develop the skills is presented. Section 4 shows the use of the framework. Finally, in section 5 conclusions and future developments to be carried out, are described.

2. FRAMEWORK OF THE SEQUENCER AGENT

The AD architecture, as was stated before, defines the skills sequencing as a task that some deliberative and complex skills can perform. Each skill that contains a sequencing process can vary in the same aspects or requirements. Among some of these requirements it can be highlighted the requisite that the deliberative skill Navigator has, where, besides executing the plan needs to supervise its correct execution. In some complex skills, calculations and intermediate notifications must be supervised too.

The software architecture in charge of the sequencing was designed being based in the Framework development concepts proposed by Fayad (Fayad et al, 1999). For its design the process parts which not vary where defined, that is, behaviors that in any sequencing process are invariable. These behaviors are called frozen points or rigid points of the sequencing process (frozen spot). Also, those behaviors that could be variable where established and, being based on these, the methods that each component needed to modify or alter its behavior were defined. These methods were called the environment variable points. (hot spot).

The software architecture for the sequencing process (Rivero, 2003) was designed being based on the theory fundaments in which the language design was based. This language has a semantic based on the discrete event systems, modeled through a Petri net. In this language each execution unit is called process. Each sequence process has associated a set of transitions in charge of sensing the environment’s changes and, based on the detected changes, the next process to be executed is chosen. This is the invariant part of the sequencing engine. The set of transitions is managed by the transition agent.

Figure 1 shows the sequencer agent class diagram. In this design each component has a well defined interface which shows its functionality.

![Fig. 1. Class diagram of the components of the sequencer agent.](image)

The sequencing environment was divided in two components:

- Plan execution components: this component is in charge of starting and executing the plan, as well as managing the errors.
- Data and events managing component: These components allow to associate to a process the necessary data objects for the application and the data objects that will manage the events generation of that application.

3. COMPLEX SKILL GENERATOR FRAMEWORK

Skills are all perception and action capacities of a robot. In AD architecture, each robot skill represents capacity or behavior of the robot and it is implemented through a specific class, (Boada, 2002) which has associated an instance of the data component and of the event manager component. Besides, if the skill is a complex skill, it has associated a component of the sequencer agent that is in charge of managing the sequence.

Figure 2 shows the environment class diagram for the complex skill generator. Almost all the classes are based on design patterns (Gamma, 1995), the
RobotActiveObject is based on the design pattern known as active object and it allows each skill to execute itself in an independent execution thread. This class has two methods: one to activate the skill and the other to deactivate it.

![Fig. 2. Class diagram of the components to build a complex skill.](image)

ComplexSkill classes inherit from the ObjectActive class, and it is an abstract class. The new class to be developed must inherit from it and through the hot spots, it must adapt the skill to the new behavior. The variant methods, declared as virtual, allow the skill adjustment to new specifications. The onStartup method is used to established start up values, conditions or actions of the skill. The onEndActivity method allows to established actions to be carried out when the execution sequence ends.

Another method defined in the SkillComplex class is the notifyEvent method. Its goal is to allow the sequencer agent to notify the events that are generated during the sequence execution. This method internally executes the virtual method onNotify, which allows software developers to program actions related to an event notification. In the next section an example which shows the notification mechanism will be given.

In the AD architecture it is established that the automatic level communicates with the deliberative level or with higher levels automatic skills using events. These events allow to notify what is happening during the skill execution. For this reason, the ComplexSkill class inherits from the RobotEventManager class, which is a publisher-subscriber pattern implementation. This class is in charge of notifying the events generated by the skill.

The software architecture was designed considering a distributed environment. Each skill has a server which allows to take care of the clients request. It is not exactly a client-server application, because a skill is server and client to other skills. Each skill server is considered as a CORBA object that represents an specific skill and its components. The RobotModuleComplex class, is an abstract class and it is in charge of creating generic servers of the skill and also in charge of its registration in the mentioned service.

The classes in charge of the servers creation are shown in figure 3. The ObjectContainer class, based on the container pattern, allows to store the skills Interoperable Object Request (IOR), data components and sequencing agent components. This fact allows another skill to access the components that have defined public accessing methods, for example RobotSkillData component.

![Fig. 3. Class diagram for server generation](image)

Finally, the RobotModuleComplex inherits from the factorySkill class, which is an implementation of the Factory pattern. Within it, each components creation method is defined. They will be stored in the container. This fact allows to increase the system’s flexibility, because it is specialized in some complex skill components, overwriting the methods that vary. It will only be necessary to overwrite the component’s creator method when generating an instance of this component in order to insert it in the object container.

The steps to be followed when developing a complex skill are listed below:

1. Specify and analyze the execution sequence, generating the sequence’s file.

2. Create a class with the same class name as that of the sequence and that is inherited from the ComplexSkill class. If a specific behavior needs to be adapted, the following methods must be overwritten: The onStartup method for actions when the skill is started, the onEndActivity for actions when the skill ends and the onNotify method to establish the skill events.

3. Overwrite the class execState methods of the sequencer if it is desired to carry out actions during the sequence execution (OnInit, OnExec and OnEnd methods).

4. Create a skill server inheriting from the RobotModuleComplex class and overwriting the Create method of the adequate component.
4. EXPERIMENTAL RESULTS

The framework development was built in C++ using CORBA (OmniORB 3.0) [14] and Linux as OS. The names service is used to find other CORBA components. The experimental results were carried out in a RWI-B21 robot, equipped with a vision system. The robot has two on board computers; one for image acquisition and processing and the other to execute the motion commands and obtain data from a laser and other robot sensors.

The carried out tests show the steps to follow during the development of complex skills. The first one of them shows a low level complex skill that only uses basic skills. These new skills will be used to develop a new higher level complex skill, as it is shown in the second experiment.

4.1 Corridor Following Searching for a Door on the Right Hand Side (“CorriDoorR”) complex skill.

This skill allows the robot to move along a corridor searching for a door on the right hand side. If a door is found, the robot approaches it generating the door found event (“DoorFound”). In the case where no door is not found and a corridor end is found or the time while the robot is moving overcomes a established value, the door not found event is generated (“DoorNoFound”).

To develop the complex skill, the first step is to specify the sequence of simple skills evolved, by a graphic environment or by a specification language. Figure 4 shows the sequence of simple skills and events used in the complex skill Corridor Following Searching for a Door on the Right Hand Side in a Sequential Function Chart. List 1 shows this sequence using the sequencers agent own language.

The rules that this agent must evaluate from the events generated by the skills associated to the transitions are stored in the file named CorriDoorRule.

Fig. 4. Chart with the sequence of simple skills used in the Corridor Following Searching for a Door on the Right Hand Side complex skill.

List 1. Sequence of Corridor Following Searching for a Door in the Right Hand Side complex skill in the sequencer’s agent own language.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Condition/Value</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td><code>/%DoorFound=YES / 0 / A1</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td><code>/%CorridorEnd = YES ! Time&gt;150 / 0 / A2</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td><code>/%Rotate=YES / 0 / A4</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td><code>/%Approach=YES / 0 / A3</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td><code>/%CorridorEnd = YES ! Time&gt;150 / 0 / A3</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the CorriDoorAction file actions to be performed, are:

- R1 / `%DoorFound=YES / 0 / A1`
- R2 / `%CorridorEnd = YES ! Time>150 / 0 / A2`
- R3 / `%Rotate=YES / 0 / A4`
- R4 / `%Approach=YES / 0 / A3`
- R5 / `%CorridorEnd = YES ! Time>150 / 0 / A3`

The second step is to create a new class named CorriDorrR_i, which inherits from the ComplexSkill class. In this new class, the method onNotifyEvent is overwritten to register the event generated during the skill performance (DoorFound or DoorNoFound) and then being able of notifying it to the other skills registered in the Event Manager.

In the second step, the corriDoorRExec_i, inherits from the execState_i, class. In this new class the methods onExec() and onEndExec() are overwritten. The onExec() method task is to verify that the event DoorFound was generated during the sequence execution. In the same way, the onEndExec method task is to indicate in which hand is detected, in order to notify it to the corriDoorR skill through the notifyEvent event.

Finally, a class named corriDoorRServer_i is created. This new class is the server of the CorriDoorR skill. In this class the createComplexSkill() and the createExecState() methods are overwritten allowing the IOR of the new implementations to be stored in the container of the module objects. Figure 5 shows the robots trajectory using this complex skill when traveling the corridor. When the robot detects a door on the right hand side, it stops.
4.2 Contour Following Searching for a Door in the Front ("CountDoorF") complex skill.

This skill allows the robot to move following an object contour and searching a door placed in front of it. If a door is found, the robot approaches it generating the door found event ("DoorFound"). In the case where the door is not found and an the end of the corridor is found or the time while the robot is moving around the contour overcomes a established time, a time out event is generated ("Time_out"). The list 2 shows this sequence using the sequencers agent own language. The development of this skill is the same above.

List 2. Sequence of the Contour Following Searching for a Door in the Front complex skill in the sequencers agent own language.

The rules file stores the following rules:

#1 { ContourFollow()
    [LookObjectLaser(0.0):(ContDoorRule,R1)]
    [ContourFollow()):(ContDoorRule,R2)]
}

#2  {  ApprDorr()
    [AppDoor()):(CorriDoorRule,R3)]
}

#3 { STOP }

START 1

The actions file stores the following actions:

A1 / 1 / #2
A2 / 1 / #3

Figure 6 shows the trajectory followed by the robot using this complex skill.

4.3 Higher complex skills: Go to Lab complex skill. ("GoLab").

In this test, how to build complex skills using lowest level complex skills is shown.

This skill makes the robot move along a corridor searching for a door on the right hand side. If the door is found, the robot crosses it and moves following the contour searching for a door in front of it. If the door is found, the robot goes into the lab and accomplish it task, notifying it with an event. On the other hand, if one of the doors is not detected, a failure event is generated.

Figure 7 shows the sequence of simple and complex skills and events used in the complex skill GoLab. List 3 shows this sequence using the sequencers agent own language.

List 3. Sequence of GoLab complex skill in the sequencers agent own language.

The rules file stores the following rules:

#1 { CorriDoorR()
    [CorriDoorR()):(GoLabRule,R1,R2)]
}

#2  {  CroosDoor()
    [CroosDoor()):(GoLabRule,R3,R4)]
}

#3  {  CountDoorF()
    [CountDoorF()):(GoLabRule,R5,R6)]
}

#4  {  CroosDoor()
    [CroosDoor()):(GoLabRule,R7,R4)]
}

#5 { STOP}

START 1

START 1
The actions file stores the following actions:

In this case, in the second step, a new class named GoLab was created. The onStartup method was overwritten with the new initialization values. The onNotify method was overwritten too in order to establish the events generated by this skill.

Finally, the class GoLabExec is created. The onExec method was overwritten to count up the crossed doors and the onEndExec method is overwritten to notify a failure when getting in the lab.

Figure 8 shows the trajectory followed by the robot using GoLab complex skill.

| R1 / %CorriDoor = FOUND / 0 / A1 |
| R2 / %CorriDoor = NO_FOUND / 0 / A2 |
| R3 / %CrossDoor = CROSSED / 0 / A3 |
| R4 / %CrossDoor = NO_CROSSED / 0 / A2 |
| R5 / %CountDoor = FOUND / 0 / A4 |
| R6 / %CountDoor = NO_FOUND / 0 / A2 |
| R7 / %CrossDoor = CROSSED / 0 / A2 |

A1 / 1 / #2
A2 / 1 / #5
A3 / 1 / #3
A4 / 1 / #4

In this case, in the second step, a new class named GoLab was created. The onStartup method was overwritten with the new initialization values. The onNotify method was overwritten too in order to establish the events generated by this skill.

Finally, the class GoLabExec is created. The onExec method was overwritten to count up the crossed doors and the onEndExec method is overwritten to notify a failure when getting in the lab.

Figure 8 shows the trajectory followed by the robot using GoLab complex skill.

6. CONCLUSIONS

An Agent-based framework to develop complex skills has been successfully used. Several complex skills have been built using this framework, in a very simple and easy way and in a short period of time. The use of an agent to select the following process helps developing sequences, thus increasing its flexibility. A graphical interface intended to define sequence plans and to store them in a database is under development as a net implementation class. In the future, other components shall be added to the framework. These components will evaluate the expressions when using of rules based on more complex reasoning methods.

7. ACKNOWLEDGMENT

The authors gratefully acknowledge the funds provided by the Spanish Government through the MCYT projects TAP1999-214 and DPI2002-00188.

REFERENCES


