JERRY - A System for the Automatic Generation and Execution of Plans for Robotic Devices: The Case Study of the SPIDER Arm *

A. Cesta, P. Riccucci, IP-CNR, National Research Council, Rome, Italy
M. Daniele, P. Traverso, IRST, Trento, Italy
E. Giunchiglia, M. Piaggio, DIST, University of Genoa, Italy
M. Schaerf, DIS, University of Rome "La Sapienza", Italy

Abstract

This paper describes JERRY, a system which supports the interactive design, planning, control and supervision of the operations of autonomous systems in a space environment. The aim of JERRY is to provide a high level of autonomy still retaining the possibility for the user to monitor, control and override potentially autonomous operations in a flexible way. JERRY is composed by a set of tightly integrated specialized subsystems, which have been designed to perform effectively and efficiently their specific tasks, and, at the same time, to be open to the interaction with the user and among each other. This results in a system with a potential high level degree of autonomy, but which can still be controlled and guided through interaction.

JERRY's architecture and underlying ideas have been tested and made operational for monitoring and controlling a SPIDER robotic arm operating in an indoor environment very close to the payload tutor experiment described in [5].

1 Introduction

The recent development of space autonomy results in a set of novel problems related to the integrated work of human beings and robotic devices in space missions. From the one side, the increasing complexity of the services requested to robotic devices results in a need for more and more sophisticated and autonomous systems. From the other side, a relevant aspect of space missions involving humans is their possibility to maintain a level of control over autonomous robotic devices (see for example [7; 8]). This is due to quite a number of factors, such as the possibility of a completely unexpected event that may require the humans to override the robotic autonomy; or the long run psychological effects for the human of living in an environment in which everything is "externally directed and operated".

In this paper we describe JERRY, a system which supports the interactive design, planning, control and supervision of the operations of autonomous systems in a space environment. The aim of JERRY is to provide a high level of autonomy still retaining the possibility for the user to monitor, control and override potentially autonomous operations in a flexible way. JERRY is composed by a set of tightly integrated specialized sub-systems, which have been designed to perform effectively and efficiently their specific tasks, and, at the same time, to be open to the interaction among each other. This results in a system with a potential high level degree of autonomy, but which can still be controlled and guided through interaction. The set of tools provided by JERRY, all together, have been developed to provide the following main features:

Modularity: different kinds of tasks, which are intrinsically complex and require special purpose capabilities, are handled by independent and highly specialized sub-systems.

Autonomy: the system is able to carry out tasks without a continuous and detailed user supervision, by enabling the specialized sub-systems to exchange data autonomously and to perform their own tasks automatically.

Interactivity: the system provides the user with the ability to inspect and direct every step of a system operation, via specialized sub-systems designed as "open systems" which can satisfy different kinds of user's requests.

Flexibility: the system can be reconfigured to fit several different robotic systems and environments of interest, by allowing for the possibility to flexibly specify the application domain, to require different kinds of services to the specialized subsystems, and to exchange data with different modalities.

Adaptability: the system can be adapted to work

* Corresponding author: Amedeo Cesta, IP-CNR, Viale Marx 15, I-00137 Rome, Italy, fax:+39-06-824737, e-mail: cesta@ip.rm.cnr.it.

Proc. Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space, 1-3 June 1999 (ESA SP-440)

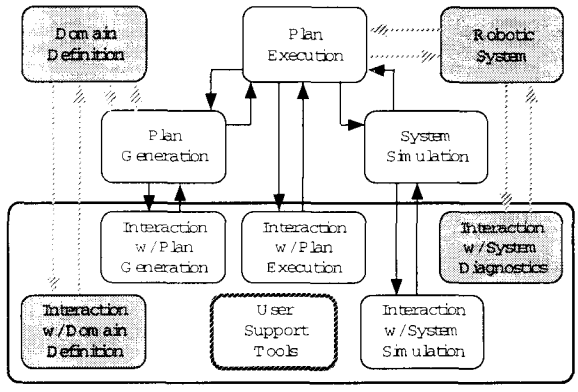

Figure 1: Structure of the System

at various levels of task specification detail and can support different user expertise.

JERRY is composed by four inter-connected modules, called the User-System Interaction module (developed at IP-CNR), the Planning module (IRST), the Execution module (DIST) and the Simulator Module (DIS). The Planning, Execution and Simulator modules are highly autonomous but open sub-systems, which can work at various levels of interaction. The planning module generates high level plans of actions to be executed; the execution module translates them into lower level programs and monitors executions; the simulator module provides a graphical and interactive simulation environment.

A main characteristic of JERRY is the provision of a flexible interface, through the User-System Interaction Module, which allows for different levels of interaction. It allows to access data and control the behavior of highly automatic systems by providing either high level specifications of what has to be achieved or detailed constraints on how the task should be performed. For instance, the user can request the planning module to generate automatically a high-level plan which achieves a high-level specified goal, or can direct the planner by imposing constraints on how to generate the plan. Analogously, the user can request the execution module to generate automatically the low level program corresponding to a plan, or can direct the execution module by imposing constraints on how the low-level robotic plan has to be generated. Finally, the user can directly monitor the execution of the program by looking at the simulation, or can directly interact with the simulator and specify the final destination to be reached. Such a high-level degree of interactivity between the user and the robotic device has been obtained via a "client-server configuration" in which the User-System Interaction Module is central to the system and can request different services from the other modules.

The structure of the system is represented in Figure 1. In this figure, the planning, execution and simulator modules are visible in the top part, while the interaction module (with the sub-modules acting as interfaces with one of the other modules) is the "big box" at the bottom. The "Domain Definition" box represents a module that allows the user to specify the domain considered, and is currently part of the simulator. The "Robotic System" box represents the real robotic device. The solid arrows represent a flow of information, while the dotted arrows represent a still missing connection. For example, the dashed arrow between "Domain Definition" box and the interface, means that currently the user can specify a domain not through the interface, but only interacting directly with this module.

Finally, JERRY has been developed as part of an ongoing and more ambitious project funded by ASI, the Italian Space Agency. In this application, JERRY provides its functionality to different kinds of users which have to design, control and monitor a SPIDER Robot Arm performing quite complex tasks, e.g., the set up of several kinds of experiments in a space workcell. Even though the project is still running, a first prototype is already working and available for experimentation. The prototype produces plans for problem in a scenario which is quite close to the payload tutor experiment described in [5]. In this scenario, e.g., the SPIDER arm is supposed to extract a tray from a shelf, fix it to one out of two tables and then automatically perform experiments moving objects contained in the tray. As far as the whole project is concerned, the functionalities of the whole system will be those of JERRY, integrated by the services provided by a module for diagnosis [9], a module for the visual interpretation of arm's activities [1], and a module (see [4]) responsible for supervising the arm in a outdoor environment similar to that described in [6]. See the corresponding papers (in this volume) for more information on any of these additional modules.

In this paper, we first provide a global overview of JERRY by describing its high level architecture (Section 2). We then describe the main features of each subsystem: the user interface (Section 3), the planning module (Section 4), the execution module (Section 5), and the simulation module (Section 6). Some conclusions end the paper.

2 JERRY's Architecture

JERRY can work at two levels of interactions that are targeted to two typical users of space robotic devices: the "programmer-level" contains functionalities offered to the robotic system operator; the "user-level" deals with activities performed by on-ground scientists or payload operators. At the programmer-level, the user can program the behavior of the device using its typically low-level interface language,

e.g. the language (called PDL2) currently used to control the SPIDER arm. A typical PDL2 instruction is "MOVE LINEAR TO `point-in-space`", where `point-in-space` is a 6-tuple of real values. This level of interaction is adequate for an experienced user. Nevertheless, programming complex tasks at this level may be very difficult for a user which has no experience with the programming language, e.g. PDL2. Moreover, low-level programs can be hard to maintain and re-use. For this reason, interaction at the user-level provides also non experts (e.g. scientists) with the ability to specify robotic tasks. Such users do not need any knowledge of the underlying physical structure of the robotic device (e.g. of the degrees of freedom of the arm) or of the physical scenario (e.g. of the exact position in space of the objects). A typical high-level instruction is "GET OBJECT `object-name`".

Operationally, the two interaction levels reflect two working modalities:

user-drives-system-supervises: in this modality an expert, knowledgeable of the underlying robotic device and mission interacts with the system by describing the mission in the robotic device interface language. The mission is encoded as a low level plan which is directly executable by the execution module.

system-drives-user-supervises: in this modality the user (even a non expert, e.g. a scientist) fixes the goal in a high level specification language. The high level specification cannot be executed directly. The system generates automatically executable low level programs. This is achieved in two steps. First, the planning module generates a set of high level actions which have to be executed in different situations and which are guaranteed to achieve the goal. Then the execution module, for each high level action, generates a corresponding sequence of low-level actions in the robotic device interface language (e.g. PDL2). Independently from how the low-level plan is generated, the execution module is responsible for its execution, and for the monitoring of the behavior of the robotic system. At each step of the execution process, the user can be prompted for validating the high-level action to be executed, or, if required, the current low-level program.

The resulting architecture is highly modular and configurable: the system can be configured to work at different levels of automation (e.g. depending on the activity performed by the planning module) and the user has the possibility to flexibly access data manipulated at different levels of detail (e.g. data at the execution or at the planning level). The interface can be set to be used by users with different experience (programmers or scientists) and can also be adapted

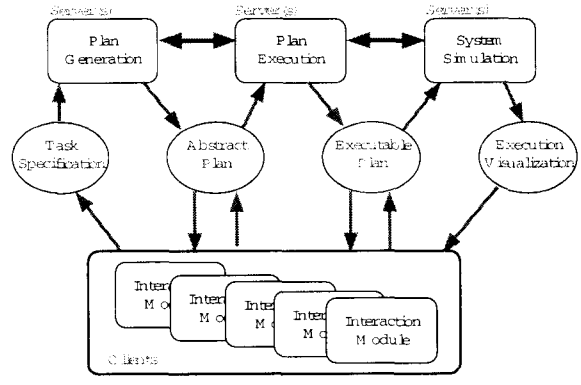

Figure 2: JERRY's Current Architecture

to different input devices (e.g., driven entirely from mouse or touchpad, entirely from keyboard, or, possibly, from custom input devices).

A first version of the demonstrator has been fully implemented, is available for inspection, and is currently under development to improve its general performance and to enrich the services offered to the user. This demonstrator (whose architecture is represented in Figure 2) is based on a client/server architecture in which a client interface service is able to continuously interact with the planning, execution and simulator modules. This has involved the development of specialized protocols that allow each interaction module to safely exchange data with the three servers through point-to-point communication. Current protocols are deliberatively designed to be very simple to minimize the overhead of communication between modules and to quickly arrive to a first integration.

3 Interaction Module

The role of software systems like JERRY is to allow different users to employ complex robotic devices while preserving the levels of responsibility that users have in their working contexts. Both the user-level and the programmer-level preserve the usual working activity, but offer a number of additional functionalities that allow the users to focus on strategic and decisional tasks and to delegate repetitive or very difficult tasks to the interactive planning software.

The JERRY Interaction Module consists of a Graphical User Interface endowed with the following functionalities:

- Task oriented help.
- Problem specification targeted to the planner domain representation language.
- Inspection of high-level plans: a rather simple representation of the plan returned by the planner is shown and the possibly of inspecting the representation of single plan states is given.

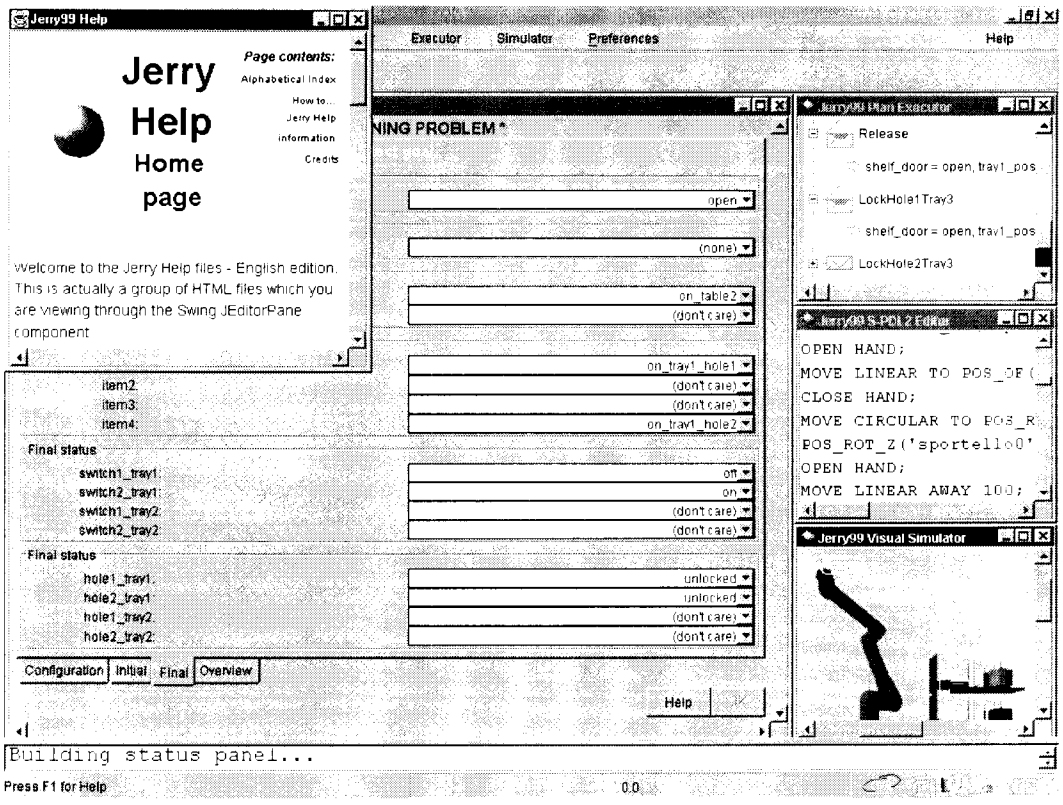

Figure 3: JERRY Interactive Module

- Inspection of plan compilation: the low-level code produced by the plan compilation and execution module is shown to the user.
- Robotic device simulator visualization.

The current look of the Interactive Module is shown in Figure 3. In the Figure we can see (*i*) the Help window (top-left) that is designed as a separate entity; (*ii*) the planning problem specifcation window (main window below the Help window); (*iii*) the plan current in execution (top-right); (*iv*) the PDL2 code corresponding to the action being executed (middle-right); and (*v*) the execution of the plan coming from the simulator (bottom-right). The size of the 4 windows corresponding to point from (*ii*) to (*v*) are interconnected and vary according to the user current focus of attention that is always contained in the main window.

According to the subdivision made between the "programmer-level" user and the "scientist-level" user, the tasks allowed to each level have been defined. In the current implementation of the "user-level interaction" the users can: (*i*) get acquainted with an operating environment; (*ii*) define specific parameters of the scenario (e.g., decide the number of trays in an experiment); (*iii*) specify the goal he want to achieve and the constraints to satisfy in achieving it; (*iv*) ask the planning module to determine the set of actions (the plan) that achieves the goal; (*v*) display and comment the resulting plan; (*vi*) activate plan execution. Special attention has been dedicated to automatically checking the consistency of commands selected by the user and in offering explanation facilities for non-expert users. The "programmer-level interaction" offers: (*i*) the possibility of creating robot programs directly using the robot language, (*ii*) the choice of having the planning and execution mechanisms that work as background help of the programmer; (*iii*) the possibility of experimenting different operational situation offering a choice among alternative input modalities. The possibility of customizing the interaction modality is relevant for experimenting on-flight use of the programming ability. In is worth observing that being the Interaction Module configured as a client it is possible to serve multiple users at the same time each of them interactiong with personalized funtionalities.

An implementation in Java (compatible with JDK 1.2) has been realized and is currently tested for improvements.

4 Plan Generation

The Planning Module developed on top of the MBP system (Model Based Planner) [2; 3], receives in input from the Interaction Module a high level specification of the task to be performed (called goal).

The goal is a high level description of what has to be achieved. It does not detail how the task should be performed. The Planning Module generates automatically a plan of actions which achieves the task specified by the goal. The plan of actions is the output which can be passed, through the Interaction Module and possibly under control of the user, to the Execution Module. A typical plan synthesized by MBP looks like the following:

```
Get object Y.
if this action succeeds,
 then put Y on experiment tray Z,
 otherwise get object Y1;
...
```

Both the goal (the high level specification of the task to be performed) and the plan of actions (the sequence of operations to be executed to achieve the goal) can be specified and inspected by the user interface. The user-level specification of the problem is translated into the representation language of the planning module. The planning module returns to the user-interface a representation of the plan which associates to each operation in the plan a description of the situation (the state) which should be reached after executing the operation.

A main characteristic of the planning module is that it is an open system, i.e. each of its operations (e.g. plan search) can be inspected, controlled and guided by the user. This fact opens up the possibility to provide a planning functionality which supports a "user-centered operation mode" for JERRY, in which the planner interacts flexibly with the user interface module. The user, beyond asking for a goal to be satisfied, can ask the planner for different services, e.g. show all the plans which satisfy a goal, select one of them, query the planner about the possible effects of the execution of plans, re-use existing plans, ask the planner to validate a user defined plan, inhibit some plans, query the planner about the current state of the execution in terms of high-level actions. This "user-centered" modality requires a design of the planning module which is different in philosophy wrt current state of the art planners. The planner is no longer the automatic generator of solutions, it becomes a system which exploits its automatic generation capabilities to support the user to find the right solution and is flexible enough to adjust its plan generation activity to different user requirements.

Another characteristic of the planning module is that MBP returns "safe plans", i.e. plans which are guaranteed to achieve the goal in spite of nondeterminism. For example, MBP is able to find a safe plan (assuming that one such a plan exists) even in the case in which some actions may fail (e.g. because of some malfunctioning of the devices) or in the case some action is no longer executable (e.g. because some of the actuators is broken).

For efficiency reasons, the planning module has been written in C. To improve its portability, the standard ANSI has been followed.

5 Plan Compilation/Execution

The Plan Compilation/Execution module is responsible for transforming a high-level, user-oriented abstract plan into a sequence of low-level, machine-oriented execution plan. In more detail, the Plan Compilation/Execution module receives in input from the interface an arbitrarily long sequence of actions to be performed, and generates a sequence of actions (a "program") that the robot can directly execute. For example, in the case of a robotic arm, the program corresponding to a $move(o,l)$ ("move object o to location l") looks like the following sequence of instructions

```
move_near <pos_o> by 800;
open_hand;
move_linear <pos_o>;
close_hand;
move_near <pos_l> BY 800;
move_linear <pos_l>;
open_hand;
move_away 1200;
```

where `<pos_o>` and `<pos_l>` are six tuples of real numbers specifying the positions of the object and of the location respectively.

In any case, the sequence of actions given to the execution module does not need to correspond to a complete plan. Instead, the user can (*i*) break a plan as given by the planning module into blocks of planning actions, (*ii*) require the compilation of all or some of the blocks, (*iii*) validate the execution program corresponding to a program, or (*iv*) ask for an execution program differing from the proposed one.

As for the planning module, the execution module is an open system in which the parameters affecting its behavior (e.g. the availability of a given low-level action) can be inspected, controlled and eventually modified by the user. For example, the user can inhibit the execution module from using a certain low-level action because it involves some dangerous or unavailable move for some joint. As above, this fact opens up the possibility to provide a "user-centered operation mode" for JERRY, in which the execution module interacts flexibly with the user interface module.

A Java (compatible with JDK 1.2) implementation of the execution module has been realized, and is currently tested for improvements.

6 Robotic System Simulator

The simulator allows for a 3D representation of a robotic arm in a given working environment. The

simulator is composed by three parts:

- a user interface which allows the user to examine the scene and to interact with it by suitable commands;
- an interface that allows the user to define all the objects in the scenario;
- and interface that allows the user to define the robot employed to manipulate the objects in the scene.

Currently, the simulator has been specialized with knowledge of two domains: the first is close to the external robotic experiment described in [6]; the second domain resembles the internal payload tending described in [5]. In both cases, the robotic device is the SPIDER arm.

About the user interface, the operator may:

- observe the evolution of the scene on a screen, both by looking at the arm's movements and information of the specific values of the various variables controlling the arm,
- interact with the robot, e.g. by specifying a position to be reached, or
- control the robot, by writing a PDL2 program which can be executed.

Finally, the simulator has been written using the Java language, the Java 3D library, while some VRML files specify the geometry of the objects in the scenario. The simulator is therefore a Java application that does not depend on the particular external browser used.

7 Conclusions

This paper describes JERRY, a system for the automatic generation and execution of plans for robotic devices, and briefly reports about the case study of the SPIDER arm. The main feature of the system is the high-level of interaction that the user can decide to have with the system. This level of interaction is critical in the context of spatial missions, where (*i*) unforeseen emergencies can happen, and (*ii*) still the mission has to proceed, possibly under the humans' supervision.

JERRY has been designed to be a flexible, open architecture. Care has been taken in order to distinguish the domain-dependent from the domain-independent tasks in order to minimize the customization efforts. JERRY's architecture and underlying ideas have been tested and made operational for monitoring and controlling a SPIDER robotic arm operating in an indoor environment very close to the payload tutor experiment described in [5].

Acknowledgments

This research is supported by ASI (Italian Space Agency) as part of the project "Un Sistema Intelligente per la Supervisione di Robot Autonomi nello Spazio". Special thanks to the other participants in the project for the many fruitful discussions and the technical help.

References

[1] A. Chella, S. Gaglio, D. Guarino, and I. Infantino. An Artificial High-Level Vision Agent for the interpretation of the Operations of a Robotic Arm. In *Proc. 5th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS99*, ESTEC, Noordwijk, NL, 1999.

[2] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for AR. In *Lecture Notes in Computer Science*, volume 1348, 1997.

[3] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press, 1998.

[4] A. Dell'Arciprete, G. Rossi, A. Finzi, F. Pirri, and M. Schaerf. A System Integrating High and Low Level Planning of Complex Tasks with a 3 Dimensional Visualizer. In *Proc. 5th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS99*, ESTEC, Noordwijk, NL, 1999.

[5] S. Di Pippo, G. Colombina, R. Boumans, and P. Putz. Future Potential Applications of Robotics for the International Space Station, Robotics and Autonomous Systems. *Robotics and Autonomous Systems*, 23:37–43, 1998.

[6] F. Didot, J. Dettmann, S. Losito, D. Torfs, and G. Colombina. JERICO: A Demonstration of Autonomous Robotic Servicing on the MIR Space Station. *Robotics and Autonomous Systems*, 23:29–36, 1998.

[7] G. Dorais, P. Bonasso, D. Kortenkamp, B. Pell, and D. Schreckenghost. Adjustable Autonomy for Human-Centered Autonomous Systems on Mars. In *Proc. Mars Society Conference*, 1998.

[8] JERICO Project Description. ASI Internal Documentation., 1998.

[9] L. Portinale, P. Torasso, and G. Correndo. Knowledge Representation and Reasoning for Fault Identification in a Space Robot Arm. In *Proc. 5th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS99*, ESTEC, Noordwijk, NL, 1999.