SEFFAS
Simulator Environment For Flexible Assembly Systems

A.H.J. Koning

April 1991

Abstract

This paper describes the design and implementation of a simulator forso-called flexible assembly systems. These are robot cells with multiplerobots and sensors designed to be able to produce different kinds of products.This project is a part of the Esprit Computer Integrated Manufacturing ProjectNo. 2202 - PLAT - Planning Toolbox for CIM systems. The simulator has tobe capable of interactive simulating a robot cell with multiple robotsand sensors. With simulating we mean both the emulation of the control ofthe cell, and the visualization of the simulated robot cell on workstations,so that we can monitor the (emulated) execution of commands and programs bythe cell.

Contents

1 Introduction

1.1 Background

With the evolution of the electronic computer, w ich startedin the early 1950's, came the evolution of the robot in the early 1960's.But it was not until the late 1970's that robots became cheap and versatileenough to be used on a large scale in the industry. During the years thatfollowed robot utilization increased rapidly and in the mid 1980's it wasexpected that automated assembly systems (i.e. robots) would replace thetraditional systems before 1991. But contrary to the expectations in the second half of the 1980's theincrease in robot util zation came to a halt. The reason for this wasthat although the mechanical evolution had led to robots capable ofalmost any manipulation needed, the real problems lay with the softwaredevelopment and the integration and interpretation of sensor data in theassembly process. Research in these areas has led to the insight that robotsmust not be seen as isolated units but that for successful application ofrobots it is essential that more emphasis is put on the integration aspects,leading to the overall oncept of computer integrated manufacturing (CIM). Together with the introduction of the robot came the problem of howto program it. Early techniques were based on the teach-in concept. Theprogrammer physically moves the robot through the required trajectoryduring which the motions of the robot's joints are recorded on tape orotherwise. This recording then serves as the robot program. Drawbacks ofthis method are that the robot has to taken out of the production processto be taught and that there i little control over the correctness of theprogram. With the introduction of textual representation of robot motionsthe second problem was tackled and the first robot control languages weredeveloped. These first languages were very simple and with thedevelopment of high-level general purpose languages such as C and Pascalcame more and more complex robot languages, some of them based onextensions or adaptations of general purpose languages. This however ledto the problem that programming the robot wa no longer a job for theunskilled operator but that specially trained programmers were needed. Nevertheless at the moment most industrial robots come with aninterpreter for a specialized robot language. However this still doesn'tsolve the problem of having to take the robot out of the productionprocess to program it. This programming with the help of robot's controlleritself is called on-line programming and to overcome this problem of loss ofproductivity off-line programming is introduced. Off-li e programmingallows us to write the program on a other system than the actual robotand thus is very appealing [Gini]. The idea of an off-line programming system is that we can test thecorrectness of our program without having to use the real robot. To beable to do this the off-line programming system should contain a model ofthe robot and its environment. The model should correspond closely to thereal situation in order to ensure that the program functions correctly. Part of the Esprit 623 proj ct "Operational control for robot systemintegration

into CIM" was the development of tools for productionplanning and off-line programming. Production planning and programming inComputer Integrated Manufacturing are much more complicated problems thanthe off-line programming of a single robot. This means that there agreater chance of errors in the planning and/or programming than with asingle robot. So in addition to the reasons mentioned above for a singlerobot, the fact that an error in the planni g or programming of a CIMsystem can lead to the break down of the complete production in a factorymakes that we would like to be able to simulate our CIM system so that wecan test our planning and programs before they are executed. Among the tools developed as part of this ESPRIT project is theoff-line programming and simulation package ROSI [Dillmann][Huck][Rembold] developed by the University of Karlsruhe. ROSI enables themodelling and programming of robots, parts and the robot's work cell.Part of ROSI is the simulation of the robot trajectories by a graphicssystem. In addition to programming and planning errors, it is possible thatchanges in the actual status of the robot's environment may lead tointerruptions or even a break down of the production process.This is because in the planning and programming stage we assume a certainmodel of the reality, which may only partially reflect the real world.To prevent this from happening we want our robot programs to be able tohandle these so-cal ed exceptions. A robot programming environment capableof handling exceptions has been proposed [Meijer1] and implemented [Mul]. The University of Amsterdam participated in this ESPRIT project withthe development of the "Exception Handling Model" (EHM). Exceptionhandling in this context means that the robot is able to react tounforeseen situations in a such a way that they can be handled adequatelywithout outside help, so that the robot system continues to operate.Tools have been developed to gener te and monitor the recoveryprocedures. One of these tools [Meijer2] uses an expert system to implement theexception handling. The development and testing of this tool was donewith a real robot and real sensors. The drawbacks of this approach areobvious: You can only test the system with the available sensors androbots, preventing them from being used for other purposes and until theexception handling is perfect it is possible for things to go wrong andthe set-up may get damaged. A suitable simula or would remove thesedrawbacks. Work along these lines currently continues in Esprit II project 2202:PLATO - Planning Toolbox for CIM systems. This project has as one of itsobjectives the development of a complete simulation system for CIMsystems. The University of Amsterdam will provide sensor based controland exception handling in this system. Again a simulator capable ofhandling multiple robots and sensors is needed to create a suitableprograming and testing environment.

1.2 Demands

The availability of low-cost Unix workstations with bit-map graphicsdisplays opened up a new area in off-line robot programming. Theseworkstations are

well suited for the off-line programming of robots andwhen placed in a network these workstations allow companies to have theirdepartments do product development and production planning at varioussites. Research and development is now going on in this field ofapplication, for instance the Esprit PLATO project. As a part of thisESPRIT project this gr duation project involves the design andimplementation of a simulator system suitable to run interactively onUnix workstations, and Sun workstations to be specific, and capable ofgiving a detailed simulation of an arbitrary robot cell including theintegration of sensors. As an example of this sensor integration we cantake two robots in a master-slave configuration, in which one follows theother based on information from a force sensor held by both robots. For our simulator we would like to use a st ndard windows environmentand a standard graphics library so that the simulator package can easily beported and made available on a large number of machines. As we alreadymentioned Unix is the most wide spread operating system for workstations thesedays and for Unix systems is the X-windows environment expected to become thestandard window environment. The PHIGS graphics library is the standard forthis type of visualization. So what we ultimately would like is to have oursimulator run under X-windows and making use of the PHIGS library.

1.3 Outset

At the start of this graduation project we had the result of twoprevious projects at our disposal [Roth][Mul]. This simulator called ROSor ROSI was a partial port of the ROSI package mentioned earlier,designed to run on Sun 3 workstations, and it allowed us to simulatemultiple robots. It used the PHIGS library and ran under the SunView windowenvironment. But while it fulfilled part of our demands it failed intwo important areas: It took thi simulator 20 seconds or more to drawone robot, making it more or less impossible to be used interactively.Furthermore the simulator made no provision for the integration ofsensors and neither was there a way to access the world model used fromoutside the simulator. Although this simulator did not run under theX-Windows environment, using a PHIGS library suitable for X-windows wouldenable it to do so. We started this project by looking if it was possible to use the existingsimulator and try to fi d ways to speed it up (chapter 2). When we receiveda new and considerably faster version of this simulator we decided to use itand next we concentrated on integrating the various parts of our system whichwe already had available (chapter 3-4) and extending and changing them wherenecessary (chapter 4-5). This led to a simulation environment which allowed usto realize an interactive simulation of multiple robots. The important part,sensor integration, was next and we designed and implemented a way to b abledo this (chapter 6-7). Finally we look at the result of this project and tofuture developments (chapter 8).

2 Orientation

2.1 First Effort

The simulator we had at our disposal was written in Pascal and usedan implementation of the PHIGS standard graphics library written inFORTRAN. As we mentioned earlier it ran on Sun 3 workstations. Because wenow also had the newer and considerably faster Sun 4 workstations to workwith it seemed to be a good idea to try and por ROSI to these machinesto see what increase in speed this would give us. Despite repeated efforts we did not succeed in porting the PHIGSgraphics library (FIGARO), so we could not run our simulator on thefaster SUN 4, SPARC station 1 and SPARC station 1+ machines we haveavailable. And while the PHIGS package was a commercial product it didnot integrate well with the SunView windows environment it was runningunder, so one of our options was to replace it. We took the picturequality of the ROSI sim lator as our minimum target and searched for away to get the performance we required to do real-time animation of thesimulated robot cell. This led to the following experiment:

2.2 Experiment

The first question we had to answer was whether the poor performanceof the existing program was a software or a hardware problem. We hadlittle documentation about the way PHIGS works, so this was one of thefirst things to figure out. PHIGS uses an hierarchic tree structure tostore the transformation on the objects it has to draw and keepspointers to an array with the geometric data. When given an order to drawan object it walks thought the tree, performs the transformations anddraws the object. This implicates that either the transformations or thedrawing or both are costly operations in terms of processor time. To findout where the problems really lie we did some experiments. First we needed a estimate of the complexity of the picture we want to draw.Because we use vector graphics the numbe of edges in the picture will providesuch an estimate. When we looked at the data files used by the existing programto describe the various elements in our picture, we saw that they consistedonly of polygon descriptions. One line to describe the polygon, followedby the vertices of that polygon, with each vertex on a seperate line.Since the number of vertices of a polygon is the same as the number of edgesof that polygon, we can use the length of the file in lines as the upperlimit of the number of e ges in the picture. This yields the following results: About 3500 lines (edges) for a Puma 260robot, and about 4000 for a Bosch robot. Since our application will beused to simulate cooperating robots we are looking at a picture withat least about 7,000-8,000 lines. And as we want an animated picture weprobably need to redraw the picture at least twice per second. This brings usto a total of 15,000 lines a second. The transformations are done by means of floating point matrix-vectormultiplicatio

s, so we wrote a small program that does 15,000 of thosemultiplications and timed it. It took a Sun 3 less than a second toexecute, so we can more or less safely assume that this is not thebottleneck in our application and that the lack of performance is causedby the limited speed with which the lines are actually drawn. To test this assumption we wrote and timed a little program whichonly draws lines. Since most lines in our application are pretty short,our benchmark program drew 15,000 lines wi h a length of 10 pixels.

2.3 Result

When we timed the execution of this benchmark on various systems ityielded the following results:

- SUN 3 (68020), single user: 21 seconds
- SUN 4 (SPARC), 2 users (load average 1.7): 3 seconds
- SPARC station 1, single user: 2 seconds
- SPARC station 1+, 3 users (load average 1.0): 1 second

When we compare the time for the SUN 3 benchmark with the approximate20 seconds needed to draw a real picture (of ab ut 10,000 lines) with theprogram using FIGARO, we may conclude that our benchmark gives us a goodidea of the performance which can be attained without additionalhardware. It also shows that the use of FIGARO slows down things by about30%. This seems a little more than we expected at first, but the FIGAROimplementation of PHIGS is written in FORTRAN (not the best language forthis kind of application) and not specially tuned to the SUNworkstations. From these results we can draw some conclusions: f we use a SPARCstation 1+ the simulator can draw at the required speed, but it doesn'tleave much room for overhead caused by the calculations needed to do thesimulation. Nor does it allow for the driver programs and thecommunication between the drivers and the simulator to consume any time.So from all this we conclude that we should look for a way to accelerateor drawing, or somehow reduce the amount of drawing to be done.

2.4 Open questions

After this experiment there are still some op n questions. Forinstance this experiment learns us nothing about the speed increase whichmight be attained with the use of additional hardware. Furthermore wehave no data on how fast we can do things as hidden line removal andlighting/shading to improve the detail of our simulation. Also we don'tknow how much time the robot and sensor drivers take, but some littletests show that the faster machines take very little time to do thesekinds of simple floating point operations.

2.5 Options

A already mentioned above if we want to use PHIGS, and continuealong the same lines as our existing simulation, we probably need to addhardware to attain our goal. So when we look what hardware we can use,the GX accelerator card offers up to 540,000 2D vectors per second and upto 270,000 3D vectors per second when installed in a SPARC STATION 1+[Sun1]. While these figures are of course the optimum, even under lessthan optimal conditions the GX accelerator card should provide theincrease in performanc we need. To make the best use of the card'sabilities it is advisable to use the SunPHIGS graphics library instead ofthe FIGARO implementation we currently use, since it specially optimized tomake use of the card [Sun2]. And even without the card SunPHIGS should bebetter adapted to the SunVIEW environment (a problem with FIGARO), and offerbetter overall performance since it is written in C instead of FORTRAN. Another option may be the development of a simulator that simplifiesthe picture before it is drawn to reduce to amount of drawing. When wetake the existing simulation and use it to look at a robot from adistance, a lot of very short lines are drawn, which add little to thequality of the picture. And when we use a close up view a lot of thelines which are draw are out side our field of view. This implies thatwhen we write a simulator that can decide which amount of detail to use, ie. which lines to draw to view a particular object at a particulardistance, we might achieve the performance e want, while we will stillbe able to see sufficient detail close up. Of course this has as aconsequence that we probably cannot use PHIGS and have to start fromscratch when implementing this simulator. If we are willing to settle for a simpler visualization another optionmay be to make a new representation with a 3D CAD or drawing package. Sincewe don't have such a packe at this moment we can not experiment with thisoption.

2.6 New developments

At that moment a new version of the exi ting simulator came available andwe were able to test some of the assumptions made above. This version wascompletely rewritten in C, and came with a new version of thePHIGS library, SunPHIGS. The person who wrote the original simulator nowworks on the same Esprit PLATO project at the University of Karlsruhe inGermany and they use it as the simulator for their environment[Negretto]. The simulator program now is called SIG, for Single InputGraphics. This version of the simulator runs on Sun 4 machines so that wewere able to test the increase in performance on these machines. We were also able to test this simulator on a Sparc Station 1+ with GX card.When we tested the performance of the new simulator, by having it draw therobot to be simulated, it became obvious that the predictions made above wereindeed true. When running on the Sparc Station 1+ with accelerator it takesthe simulator less than a second to draw a picture. We cannot

time thisexactly in an easy way because the UNIX time command c n not be used. Thesimulator takes quite some time to start up (about 10 seconds) and this standsin no proportion to the time required to draw to robot. So when we took these new developments in consideration, we decidedthat the best thing we could do was use the new version of the simulator.So what we now had to do was design our Simulator Environment ForFlexible Assembly Systems around the SIG simulator. When we have thisdesign we can start to adapt existing robot simulating/driving programsand mplement our own programs and sensor simulators to work in SEFFAS.Furthermore since SIG doesn't make any provisions for the integration ofsensors we will have to come up with a way to do this and implement it.In the next chapter we will show how SIG works and how we can use it asthe center of SEFFAS.

3 SIG — The Simulator

3.1 SIG — Introduction

SIG is based on a client-server model in which SIG acts as a serverfor a client program. For a program to become a client and allow it tosend commands to SIG you can link it with an object file that come withSIG. This object file contains functions which allow the client programto send commands to the SIG server. These commands can be divided intotwo catagories: Commands which affect the way SIG shows us the objectsits simulates, and commands which affect the objects in the simulationthemselves. In the first category fall among others commands to set theview position, the kind of projection used and the way objects ar draw,ie. as vector graphics or as filled polygons. In the second category fallcommands to change the world model, such as the setting the position andorientation of objects and commands to alter the configuration of a object,for instance the angles and translations of the various links of a robot.When called with the right parameters these functions allow us completecontrol of the simulator. As already mentioned SIG stands for Single Input Graphics which meansthat when used this way it can only ccept commands from one clientprogram at the same time. This is a major drawback since we want tosimulate multiple robots working independently, so we want to use separatedriver/simulator programs to do so. To be able to have multiple programssending commands to SIG we use a front end to SIG called MIG, forMultiple Input Graphics. MIG allows us to have several programs runningat the same time, all sending their commands to SIG.

3.2 SIG's place in SEFFAS

Based on the information available at that time we envisioned oursystem as follows (fig. 1):

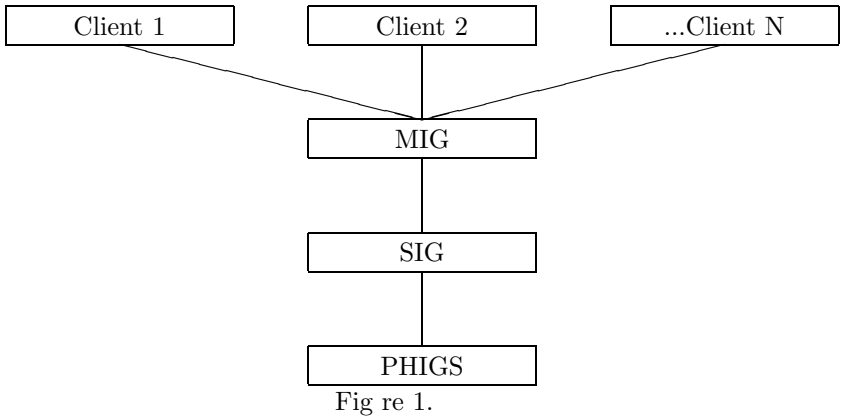


Fig re 1.

At this time we decided to first adapt an existing robot driver/simulator ARCS

(Amsterdam Robot Controller and Simulator) to work withSIG to get an idea of the possibilities of the program. After this wasdone we used the framework of ARCS to create a robot/driver simulator forthe Bosch TurboSCARA 800-4 robot we have in our robot laboratory. We willdiscuss these programs and the way they interact with SIG in anotherchapter.

3.3 SIG User's Guide

When you start "mi _server" it will start "sig" automaticallyand "sig" in its turn will start PHIGS. PHIGS is implemented as aseparate program called "phigschild" , which runs as a independentprocess and takes care of all drawing on the screen. After a short while youwill see two windows,the lower being the View Window, in which the simulator will show us theworld, and the upper window is the Control Panel which can be used toalter the way in which the simulator shows us the world. The View Windowcan be resiz d to show a smaller or greater image of the world. As youwill see Phigs always uses a square view window, whether the parentwindow is square or not. Since we are going to use this simulator with avariable number of other programs it is not possible to come up with a defaultplace and size suitable for all applications. But you can arrange the windowsin the way most suited to your situation. The control panel (fig. 2) offers a large number of buttons andsliders which can be used to alter the view pa ameters. We will now givea short tour of the most important items. First there are three slidersmarked Position X, Y and Z. The X and Y are the x and y coordinates ofthe viewpoint, i.e. the point to which we look, in millimeters. SIGstarts with a perspective view and then the z coordinate of the viewpointis 0. This point is always in the center of the view window. The Z sliderdetermines the height from which we view the world. As you already willhave noticed there is a grid drawn in the view window. This grid is 5000millimeter by 5000 millimeter in the real world, which means that eachsquare of the grid is 50 centimeters square. The sliders use the samescale, so when SIG starts with a default viewpoint of (2500,2500,1700) weare looking towards the center of the grid from a height of 170 cm.Another slider, Distance, sets the distance from the viewpoint, again in millimeters. Figure 2.

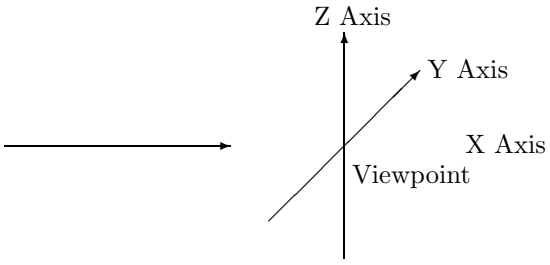


Figure 3.

Three

other sliders, Angle 1, 2 and 3, determine theangles from which we look at the viewpoint in degrees. If, for instance,we want to look at the world from above we can set slider Angle 2 to 90degrees. We can look at these angles as rotations around the three axisof th local carthesian system of the view point, as shown above (fig. 3).Slider 1 controls the rotation around the Z-axis, slider 2 the rotationaround the X-axis and slider 3 the rotation around the Y-axis. Finally theslider Focus length determines the amount of perspective, just as a cameralens. Short focal lengths give a wide view with large perspective distortion,while long focal lengths zoom in on the scene, showing a smaller portionwith less distortion. Above the sliders we find a number of switch s which allow us totoggle a number of options. One of these options, Shading, does not workat this moment, but it will be implemented in a future version of thesimulator. The Update option determines whether the simulator will redrawthe picture immediately after a change to one of the view parameters hasbeen made. When you want to change a number of parameters and you areworking on a slower machine it is wise to turn Update off, make thechanges and turn it on again, because if a complicated picture s to bedraw by such a machine it can take one or two seconds to generate aframe. If you 'slide' a slider all positions passed by the slider will bedrawn unless you have Update turned of. You can manually update the viewat any time by clicking on the Update button. The Hidden line buttontoggles hidden line removal. At the moment this only works when objectsare draw as solid polygons, and it slows down the simulator considerably. At the top of the window is a row of buttons which allow us todetermi e other, less frequently changed parameters of the simulator. Allexcept the Quit button have a menu under them which allows further choicewhich parameter to change. On the next page is an overview of this menustructure. This will be followed by a short explanation of the most importantitems. The useris encouraged to play with the view position parameters and theparameters under the view menu to create the view most suited for his orher needs and taste.

```
Update => Update
œ[7m--More--(26%)œ[m         RedrawView   => Attributes      => Default          Body
rallel                              Perspektivisch          View memory     => View load
```

The Update button we already mentioned. It has two menu items Updateand Redraw. The former draws the new si uation, while the latter restoresthe picture should it become distorted by outside influences. Next is theView button which allows you to further change the view drawn by thesimulator. For instance the Attributes item allows you to change the waythe simulator draws an object. When you select one of these items adialogue box appears into which you must enter the name of the object inthe simulation and a number which determines the new attribute of theitem. This are some example of the values you migh use:

```
Color: 1 Green  Presentation: 1 Frame    Degree of detail: 1 Frame       2 Red
```

The item Projection allows you to choose either perspectiveprojection (default) or parallel projection. If we change to a parallelprojection the Z slider becomes the z coordinate of the view point, whichnow also is no longer at the center of the view window. The bottom of theview window now is 0, and the Z sliders controls how far the 0 level ofthe grid is above that bottom. The Build button offers us a menu with some objects we can put in oursimulator's world. It was mainly meant for test purposes, since everyclient program can send the simulator commands to put objects in theworld. Finally there is the Quit button which is pretty obvious. Rememberhowever that clicking this button only closes the windows and halts theprogram. To really stop the program inte rupt it with 'ctrl-C' or 'ctrl-?'.

4 ARCS — Adaptation and extension

4.1 Introduction

When we started this project we already had another robot simulator,providing both kinematic emulation and simple visualization,called ARCS, for Amsterdam Robot Controller and Simulator. This simulatorwas developed at our university to control and simulate the OSCAR robot.This is a 6 degrees of freedom robot designed and build by the Philipscompany. It kinematic structure is much ike that of the well known Pumarobots. It has 6 rotating links, but unlike the Puma robot OSCAR has an armwithout sideways offsets. So the OSCAR can be represented as a line figureas in the following picture (fig.

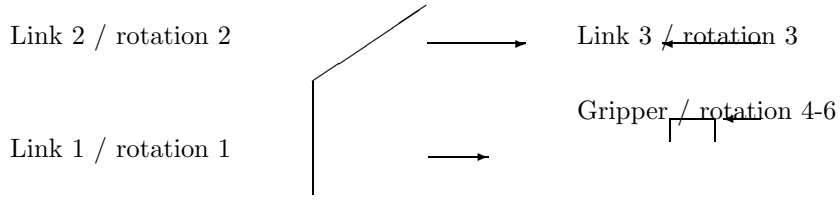


Figure 4.

4). ut(60,30)
And this was exactly the way ARCS simulated the OSCAR robot by meansof a three dimensional line model. The advantage of this type ofsimulation was the speed. Even when running on a Sun 3 workstation thespeed was large enough to do real time simulation. That was about theonly advantage because there is little to see about a line model and itonly gives you a rough impression whether the simulated move is indeedwhat you expect. But because Unix doesn't offer genuine real time controleven this part of the simulation left something to be desired.

4.2 Adapting ARCS

So we decided to start and try to adapt ARCS to use SIG for graphicsoutput instead of its own line figure. But there was a little snag.Because SIG offe s us a detailed representation of the robot it needs anfairly extensive description of the robot it simulates. And we didn'thave one for the OSCAR robot. Furthermore we didn't have a 3D CAD ordrawing program suitable to make such a description. So for the timebeing we decided to use the description of the Puma 260 robot we did have(fig.5). As already mentioned this robot resembles the OSCAR robot in itskinematic structure, but not in its dimensions. But by using thisdescription as a stand-in we woul at least be able to try out SIG with afamiliar and tested robot driver/simulator.Figure 5.SIG came with a few

demonstration client programs, but these programsonly show some robots without moving them. Adapting ARCS so as to sendcommands to SIG to place a Puma robot in the simulated world at start upwasn't a problem. Now it was time to find out how a client program couldmove the robots it had simulated by SIG. From the sources of SIG welearned that every object in the sim lation that could be moved had anumber of control values. In case of the PUMA robot these control valueswere the 6 joint angles. So by sending the simulator a command to changethese control values and then having it redraw the view we were able tomove the robot. What we needed to do next was find a point in the existing simulatorpart of ARCS where these values were known. Fortunately this was easy asthese values were passed as parameters to the procedure in ARCSresponsible for the simulation. So e could simple put the command tosend these values to SIG in that procedure and watch SIG draw the robot.At this time we kept the old simulation so we could compare it to the newone. When we tested this new version of ARCS it performed exactly as weexpected. Except for the differences in dimensions between the OSCAR andPUMA robots the movements of the robots in the old and new simulationwere the same. Furthermore the speed of the SIG simulation was close tothat of the old ARCS run on a Sun 3 worksta ion. This proved that weindeed could use SIG as a replacement of our existing simulation.

4.3 Adding a user interface

While SIG offered us a nice user interface in the form of the controlpanel, at time ARCS had nothing more to offer than a simple command line,reading its commands from ¡stdin¿. While this made it easy to execute'programs' by reading commands from file rather than from the keyboard,it wasn't very user friendly and didn't look very professional. So as wenow were altering th program anyway we might as well try to add a moreprofessional and friendly user interface. For this user interface we hadtwo candidates: First there was the SCIL interface [Kate], the result ofan earlier image processing project, which is currently being used as thefront end annex user interface for two image processing programs SCILAIMand SCILIMAGE. Secondly we could use a control panel created for use withthe driver program of the OSCAR robot. ARCS offered us the following commands: (P)oint to point move,(C)artesian straight line move, (G)rip, (U)ngrip, (F)ile read, (S)tatusdisplay and (Q)uit. We will have to implement an user interface with atleast the same functionality. SCIL stands for Standard C Interpreter Language and this is the mostimportant feature of this user interface. It was developed to offer acomplete development environment for image processing programs. Inaddition to a C interpreter SCIL also offers a menu system, on topof it, which allowsthe user to call compiled fun tions linked to SCIL and a way to enterparameters for these functions by means of dialogue boxes. SCIL alsoincludes a command expander which allows the user to call functions withtheir parameters from a command line. For our ARCS interface we want a

convenient way to call the functionsnecessary to perform the commands listed above. We don't want to use thecommand expander because ARCS already offered us that kind of interface.So we placed the commands as functions in a menu. When such a menu items is selected a dialogue box appears allowing us to enter the parametersassociated with the function. When used this way we notice that this is notquite the way SCIL was meant to be used. Dialogue boxes open for eachfunction (even if it has no parameters) and unless closed clutter ourscreen. Since is not possible to put commentary or icons in the dialogueboxes they all look alike and when a number are open this is confusing. A solution might be to put a number (or all) of the functions underthe same menu item and in the same dialogue box. This however would leadto a very large and ugly dialogue box since every parameter comes on itsown line. To put in Apple terms: 'The look and feel' of the interface isnot right for this kind of application. So we may conclude that while itis possible to use SCIL as the user interface for a robot driver/simulatorit is not the most optimal solution. It would however be possible andprobably advisable to use SCIL as the user interface for vision orientedsensor sim lators which might be developed for SEFFAS in the future. Forinstance the use of a stereo camera for object recognition and/or sensordriven robot control. At this moment we lack a programming environment suitable for multiplerobots and sensors. This falls outside the scope of this project, but SCILlooks a good candidate if it is extended with the proper routines to controland communicate with the client programs. We will return to this in the finalchapter. But even if we are to use SCIL for this p rpose we still need aworkable user interface for the immediate control of out robots. The next option we had was to 'borrow' the control panel of the OSCARdriver program. So we took the relevant source file of that program andreplaced the various function calls with our own. We did not encounterany difficulties except for the single step trace option built into theOSCAR driver program. This requires the driver program to return controlto the control panel after every single step in a move. To impl ment thisin our driver/simulator would require a complete rewrite of the movementroutines so we left this option out. It would be possible however toimplement a single step trace option in the simulator by having it waitfor a keypress after each step. But since this is not intuitive and theold ARCS did not offer this option anyway, it is not implemented. After implementing this interface and trying it out we could findonly one drawback: Because it offers all the control options at the sametime it is a bit large. While this may not be a problem when only one isrunning the screen may become crammed if two or more are active at thesame time. So the user is advised to iconify (close) any control panelthat isn't actively used at a given time. However it works and looks goodso we decided to stay with this user interface.

4.4 ARCS User's Manual

You can start ARCS by executing the program arcs after you havestarted the simulator SIG. ARCS will now prompt you to enter the mode inwhich it ill run. Enter "m" to put ARCS in master mode and be able to move the robot with the control panel. The use of slave mode will be explainedin the chapter about the World Model Manager. Next you'll have to wait afew moments while the simulator loads the description of the Puma robot.You will see the robot appear in the simulators view window and finallythe control panel will appear (fig. 6). You can now enter a destination by either moving the sliders or bytyping the desired coordinates after the Value: prompt. Don't forget toclick the Set button to set the sliders to these values. When enteringthe destination's coordinates remember that the size of the Puma robot inthe simulation is about half that of the OSCAR robot. This means thatwhen you enter the coordinates shown in the program above the Puma robot willappear to move to about (662,0,350). It was easier to leave it thisway so that if we get data files for an OSCAR robot we don't have toadapt ARCS again. At the moment the adaption of AR S is more meant as atrial to see how this combination with SIG would work out. Next you can use the Steps slider to set the number of steps in whichthe simulator will show the move. On slower machines without a graphicsaccelerator it is advisable not to use more than 20 steps. Finally youcan change the desired motion type from point to point to straight lineor back. The robot will try to execute the move to the position andorientation set by you when you click the Move button. If a position ororientation can not be reached the robot will stop and a message willappear. You can open and close the gripper with the Grasp and Detach buttonsbut at the moment there is no provision in ARCS to pick up an object evenif there is one in the simulation. See the chapter about futureextensions for more information. At any time you can move the robot toits home position in 10 steps by pressing the Reset button. Clicking onPrint info will print information about the position and orientation ofthe robot. Finally ARCS can execute off-line programs by entering thefilename and clicking the Execute button in the lower part of the controlpanel. As already mentioned in the previous paragraph the Single steptrace option doesn't work. Off-line programs consist of the commands mentioned earlier, followedby their parameters if any. For instance the following programs firstprints the current position then performs a straight line move toposition (1324, 0, 700) and orientation (0, 90, 0) in 20 steps andfinally prints the new position:

```
s  c  1324  0  700  0  90  0  20  s
```

Note that this is exactly the way you would have entered the commandsin the earlier versions of ARCS.Figure 6.

5 BOSROS — BOsch Scara RObot Simulator

5.1 Introduction

So now we had a working combination of a client and server programwhich proved that we could implement a robot simulation this waywhich suited our needs. We could now start o implement a simulation forthe Bosch TurboSCARA SR 800-4 robot which we have in our robotlaboratory. This is a powerful industry robot which among other things isbeing used for research involving vision driven robot control. It wouldbe desirable to have a simulator to use instead of the real robot whenexperimenting with this type of control. We already had the right graphical data for the Bosch robot, butthere were two problems: First of all the data appeared to be corruptedas SIG crashed when w tried to load it. Secondly the Bosch robot camewith a complete set of hardware which takes care of driving the robot.This functions as a black box which allows us to specify a goal positionand then takes care of the path planning, inverse kinematics and dynamicsto have to robot move to it. So in addition to trying to repair the datafiles we would also have to implement a black box of our own to act asthe driver/simulator program.After we succeeded in repairing the files (see appendix B) weencount red another slight problem in the fact that the Bosch descriptioncame without a gripper. Instead we used the Puma gripper as this is similarto one of the grippers we have for the Bosch. But these and the problem withthe OSCAR robot shows that it might be advisable to obtain a 3D CAD ordrawing program capable of creating files in this format or some format thatcan be converted to it. Together these files give us a quite workablesimulation of the Bosch robot (fig. 7).

5.2 BOSROS — The driver simulator

Now we had the files working it was time to implement a driver/simulator program. Because we wanted the same functionality as ARCSoffers we decided to replace the kinematic routines in ARCS with routinessuitable for the Bosch SCARA robot. There is one difference however,while ARCS was intended to study the dynamic behavior of the robot, thisis not one of the main concerns with the Bosch robot. One of the Boschrobot's main applications is the combination of robot control and vision,so p sitioning is more important than speed. And since the robot can movefaster than the simulator can draw anyway we decided to ignore thedynamics and just simulate the movement. It would be possible however toextend the simulator to include dynamics.Figure 7.ARCS

uses matrix calculations to solve the kinematics of the OSCARrobot. Instead of trying to adapt these routines it was simpler to usesome simple trigonometry to solve the kinematics of the SCARA robot. Therobot has onl 4 degrees of freedom as opposed to the 6 of the OSCARrobot (fig. 8). The orientation of is controlled by one rotation and inaddition movement in the Z direction depends on only one translation inthat direction. This means that the X,Y positioning is controlled by tworotations. This implies that we only have to solve the 2 dimensionaloblique triangle formed by the links of the robot and the imaginary linefrom the first rotation axis of the robot link to the last rotation andtranslation axis (fig. 9)
The calculation goes as follows:

```
X = x coordinate of destination  Y = y coordinate of destination  A = length of first
```

There are other ways to perform the same calculation but they don'thold any significant advantage over this ne. After the angles arecalculated this way they are corrected for the configuration of the arm(left or right bend) and the quadrant in which the destination lies. Next we implement the two ways of movement: Point to Point andStraight Line. Since the Bosch controller is implemented as a black boxwe have no access to the functions used to perform these movements.Experimentally determining them would fall outside the scope of thisproject, so we decided to implement our own functions. Point to point ovement is achieved by interpolating the starting and destination jointangles with a cubic spline, while straight line movement is achieved byinterpolation with a cubic spline along a straight line. The number of steps in which the move takes place has no relation withany real world time. It is simply the degree of accuracy with which themovement is simulated. If we were to have a faster machine we couldextend the simulation to do real time simulation.Figure 8.Figure 9

5.3 BOSROS User's Manual

Since BOSROS uses the same control panel as ARCS, see paragraph 5.4for a description of this. There are some small differences between theway BOSROS functions and the way ARCS functions. First as the Bosch SCARArobot has only 4 degrees of freedom, only the first four sliders are used(fig. 10). Setting the last two sliders has no effect whatsoever.Furthermore since we use the right files for the Bosch robot the grid ofthe simulator, which is equal to 00 mm by 500 mm, can be used todetermine the destination coordinates. Be careful however when settingthe Z coordinate. The simulator uses a Puma type gripper and BOSROS usesthe length of this gripper to calculate the translation parameter. Ourown Bosch robot is frequently equipped with a much longer gripper andthis will lead to a difference between the real and the simulated view.Again the remedy would be to create the necessary data files and adaptBOSROS. Finally in contrast to ARCS, BOSROS will refuse to start to execute amove of which the destination can not be reached. This is true for bothpoint to point and straight line moves. This is done on purpose since ifa destination can not be reached the command to go there was probably amistake and to prevent damage to objects which may be in the way of therobot the robot refuses to move, giving the human controller the chanceto check the entered coordinates. Note that both the simulator and thereal Bosch robot don't know about the floor or tab e and will happily tryto press the gripper right trough it if you tell them to do so.Figure 10.

6 World Model Manager — Design

6.1 Introduction

When we want to do a faithful simulation of a robot workstation, weneed more than just a graphical simulation of the robot and itssurroundings. For instance, we also want to be able to detect collisionsbetween the robot and the other objects in the simulation. And if we havemore robots working together, we also w nt a mechanism to coordinate theactions of the robots. Finally we want to be able to expand oursimulation with sensor simulation. So what we really want is a simulationof the entire world around our robots. The relevant data of thissimulated world must be kept somewhere and be made available to whomeverneeds it. To do this we need a so called world manager. In the nextparagraphs we will look at what would be the best design for such a worldmanager in our specific environment. In our environment w use our own programs as clients of SIG, whichinclude ARCS (Amsterdam Robot Controller & Simulator) for a OSCAR (Puma260 like) robot, and BOSROS (BOsch Scara RObot Simulator). These clientsacts as robot controllers: The user can enter commands for the robot, andthese programs will try to perform these commands. The results of theseprograms are new states of the robots, and these are be shown by thesimulator server program, SIG (fig. 11). When we want to include sensorsand we want our robots to act n information from these sensors we need away for the robot drivers to obtain this sensor information and for thesensors to obtain information about the world.

6.2 Our current system

If we want to implement a world manager it will have to work closelywith the simulator, so we had to take a closer look at our MIG/SIGsimulator to get an idea how it is organized exactly. It became apparentto us that the way we envisioned our system wasn't completely right. Thenames of the programs MIG and S G, for Multiple and Single Input Graphicsare confusing, and some might say badly chosen. Instead of one MIGaccepting input from multiple clients, each client program has its ownMIG. These MIGs all communicate with SIG. So each MIG has only a singleinput, while SIG receives input from multiple MIGs. Figure 11.At the moment a fully operational simulation system with N

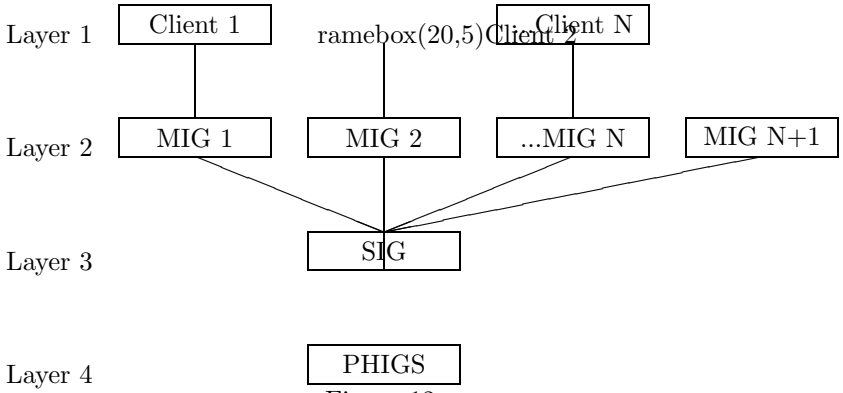


Figure 12.

clientslooks like this (fig. 12):

6.3 The current model

It is obvious that since we are simulating robots, we already have apart of our world model in the model of these robots and theirsurroundings. This model is stored in layer 3 and used by layer 4 of oursimulation system. It consists of groups of polygons to describe thegeometry of the various parts of a robot, and of transformations todescribe the relations between the parts. This data is normally kept infiles, and read in at the start of the simulation. The client programs have knowledge of the robot they are supposedlycontrolling, but this is incomplete and unstructured data. And althoughthere is communication possible between the various layers in our currentenvironment, at the moment there is no provision for use of data in thelower layers by th clients. So what we have are a collection of polygons and the relation betweenvarious groups of these polygons, which are not readily accessible.

6.4 Desires, wishes and needs

What we want is a world simulation that contains all data normallyused by a robot workstation regarding its surroundings and itself. Incase of sensor driven controllers this means that we want to be able tohave our controllers obtain data from the sensors in our world. Thesesensors will probably be simulated by eparate processes, which in theirturn also need information from the world (since this is exactly theirjob). So what we need is a world model that can be accessed by bothcontroller and sensor simulators.

6.5 Solutions

Some of the data we need is already present in our simulation, i.e.the geometry of the objects in our simulated world together with theirposition and configuration.

The problem is that we can not easily accessit. To be able to do so would require a major extension of SIG. urthermore it would drasticly increase the information stream betweenlayer 1 and layer 3. Since this data has to be present in the SIG layer we have twooptions left: Either we implement the World Model Manager as a separateprocess which keep a copy of that data, or we extend SIG to include theWorld Model Manager. If we implement our World Model Manager along the same lines as thegraphic simulator this will lead to the following situation (fig. 13):

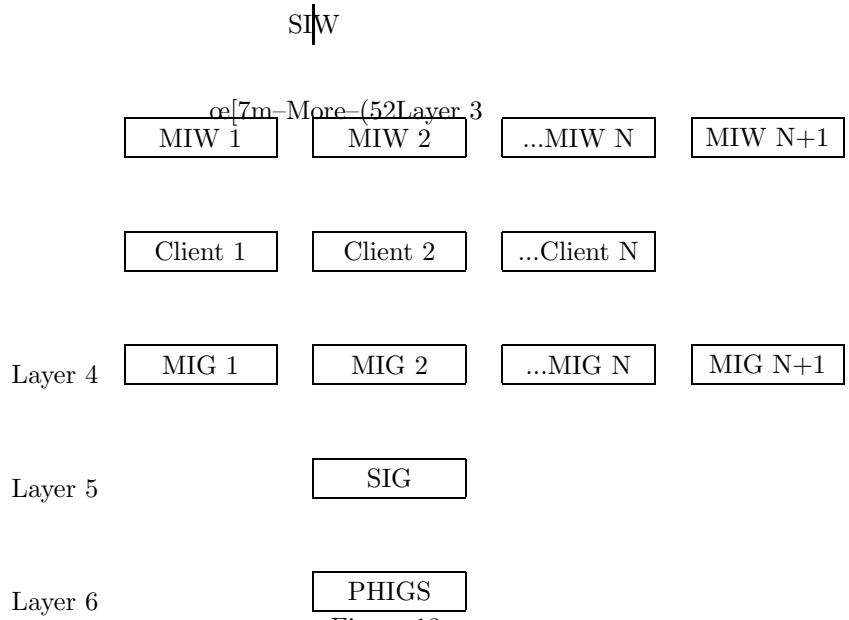


Figure 13.

If

we choose the second option there will be little change in oursystem, since there will be no new processes or communication lines (fig. 14):

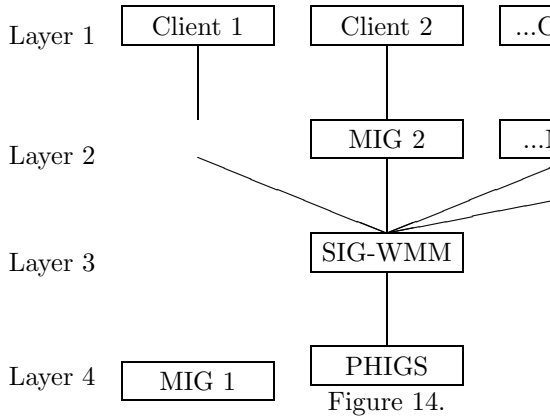


Figure 14.

6.6 Conclusions

The first option is certainly the most flexible, it allows uscomplete freedom in the way we want to implement our world model and weare able to run the World Model Manager on a different workstation thatthe one(s) we use to run the clients and the graphic simulator.Furthermore it prevents us from having to change SIG. But there are a number of substantial drawbacks to this approach. First of all it willincrease the number of separate processes from 2N+3 to 3N+5. This willincrease the memory, processor and communication overhead considerably.Secondly since we're using a large data structure we introduce morememory and processor overhead in maintaining it. Thirdly the whole systemwill become rather unwieldy to run and maintain for the user. If we expand the existing SIG program to include our World ModelManager, we do not introd ce any new processes, and the communicationoverhead will be less because we do not need to tell our world model ofchanges like new objects, positions of objects and control values ofobjects, since these will be send to SIG any way and we can interceptthem, or use the data kept by SIG. As I've already mentioned we do notneed to keep a separate copy of our object data, since we can use SIG's.Furthermore we do not introduce any new difficulties for the user. He canuse the new system the same way he wou d use the old one. With theexpectation that future users of this system won't have access to or willwant to use multiple workstations at the same time, the drawback that theWorld Model Manager runs on the same machine as the graphic simulator andcan not be run on a different one is not a problem.So if we consider all these arguments it seems that the best

thing wecan do is implement an extension to SIG to become our World ModelManager. Graphically this SIG-WMM combination will look as follows (fig. 15):

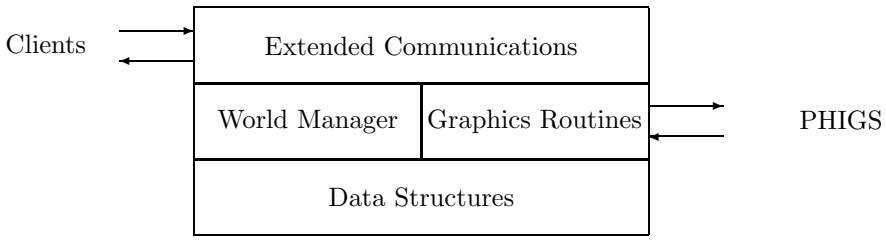


Figure 15.

In the next cha ter we will show how the World Management and theextended communications will be implemented. We will implement an examplein which we have a slave robot following a master robot, guided byinformation from a force sensor held by both robots.

7 WMM — Implementation

7.1 Introduction

This chapter will go into the technical details of the completesimulation environment and can serve as a guide when one want toimplement his own robot or sensor driver/simulators for use w th thesimulator. First we will explain how the design from the previous chapterfor a world model manager is implemented and later we will show how it can beextended to be used with other robots and sensors. What we are going to implement is a way to exchange informationbetween our world model and the client programs. The information flowgoes in both directions: The robot for instance send information aboutits position to the world model, while a sensor retrieves thisinformation and uses it to cal ulate a sensor value. This sensor value isthen passed back to the world model so that it is available for whomeveris interested in it. SIG as it is offers us a way to send information to it. We simply usethe same mechanism that is used to send commands to it, to sendinformation to our world model manager. We can then intercept thesemessages in SIG and pass them on to our own routines. There is however aproblem with the communication in the other direction. This becomesapparent when we look at the way the communication is implemented in SIG.

7.2 SEFFAS's communication scheme

The SIG part of the simulator is implemented as a program which readsits commands from 'stdin', which on the Sun workstations under Unixnormally means the keyboard. So when we would start the program called"sig" we could type in commands on the keyboard and SIG would try toexecute them. This is of course not what we would like. We want out robotdriver/simulator programs to give the commands. This can be accom lishedby using a pipe to connect the 'stdout' of our client program to the 'stdin'of SIG. However this prevents us from using more than one client programat the same time. So instead of using just SIG we use the front-end "mig_server" .When we start this program it starts SIG and connects its 'stdout' to'stdin' of SIG and vice versa. When we now start a client program, thisprograms makes contact with the mig-server be means of a communicationconstruct called a socket. This socket allows two wa communication between the twoprograms. Next mig-server forks off a copy of itself (starts a duplicateof itself) which then waits for another client program. If another clientis started and makes contact with this duplicate the whole processrepeats itself. All copies of mig-server will use the same pipes to SIG.To prevent multiple mig-servers from using the pipes at the same time aread/write semaphore (a construct used to prevent multiple clients fromusing the same resource at the same time) on the ipes. This leads to the following communications protocol: When a clientwants to send a command to SIG it first sends it to its mig-sever

whichthen tries to set the semaphore to get read/write permission on thepipes. It keeps trying until it gets permission and then writes thecommand it received from the client to the pipe to SIG. What happens nextdepends on whether the client program has set answer mode or not. If ithas it will wait until it receives an answer from SIG whether the commandwas suc essfully executed. So if answer mode is set the mig-server waitsfor a answer from SIG and then passes it on to its client. It thenreleases its read/write permission so that other clients can send theircommands. While this protocol and its implementation functions well when givingcommands and receiving immediate answers it makes it impossible for SIGto send information, for instance about changes in the world model, tothe client programs. If we want full two way communication we will haveto rewrit the entire communications routines of all the programsinvolved: sig, mig-server and the client programs. As this was outsidethe scope of this project we decided not to change the communicationroutines but to slightly alter the protocol to allow the client programsto poll the world model manager for changes in the world model whichwould affect them. We implemented a routine which intercepts messages meant for WMM andpasses them on to a routine which determines which data item or items weare inter sted in and then calls the appropriate routine to retrieve themof change them. We also had to implement a mechanism to prevent clientsfrom changing data items before all programs dependent on those itemshave taken note of the value of that items. This is because, as we areusing independently running programs, we can not guarantee that everyprogram takes his turn in a regular sequence and if we do not implementsuch a mechanism client programs will miss certain changes in the worldmodel.

7.3 mplementation aspects

We will have to confirm to the communication method offered by SIGwhich is based on strings. With these strings we will have to implementthree kind of messages: First the initial report of a client to WMM,secondly the request/inquiry send by the client, and thirdly thetransmission of data either as reaction on a request/inquiry (by WMM) oras world state information (by a client). We decided on the following structure: The string starts with "WMM:'to allow our routine in IG to recognize and intercept our command. Thisis followed by a client number, under which the client is know to WMM. Incase of a new client this client number is 0. The rest of the string isdetermined by which kind of message is to be send. In case of the initialmessage the 0 is followed by the client ID. Every client program inSEFFAS should have its own unique ID. To be able to implement a data dependency mechanism a client shouldknow on which other client programs it depends for information. T eclient ID's of these programs follow the client's own ID in the initialmessage. So in case of a sensor (ID=1) depending on the actions of onerobot (ID=2) the initial string to be send may look as follows:

```
WMM: 0 1 2\n
```

For easy construction of this string a number of constants aredefined in the file "wmm.h" (see appendix D). The ID of every client towork with SEFFAS should be defined in this file. The construction of thestring can now be done with the following s atement:

```
sprintf(string, "WMM: %i %i %i\n",          NEW_CLIENT, SENSOR_ID, ROBOT_ID);
```

This string can then be send to SIG and the answer received in thefollowing way:

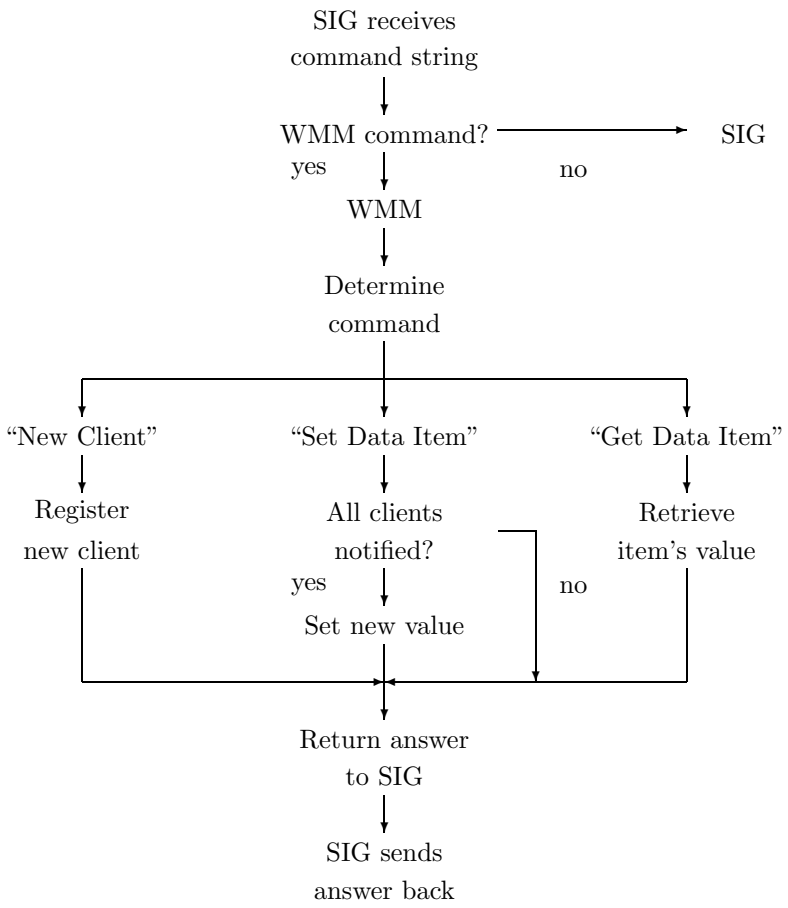
```
mig_send(string);  mig_receive(answer);
```

The answer which the client receives is also a string which in thiscase only contains an integer. This is the client number under which WMMknows the client and should be used for further communication with WMM. When a lient changes something in the world model it will have tonotify WMM. To do this a client can send a string containing the commandto do so together with the new data. For instance if we want to changethe 4 joint parameters of a Bosch robot we can build a string as follows:

```
sprintf(string, "WMM: %i %i %lf %lf %lf %lf\n", client_number,          SET_BOSCH_INFO
```

SET_BOSCH_INFO is a command number which should be defined in the"wmm. " file. Every command needs its own routine in WMM. So if we wantto keep track of the joint parameters of the Bosch robot we will have towrite a routine which reads the 4 values from the string and stores themsomewhere safe. In case of simple data items the routine necessary toretrieve this information can be implemented in the file "wmm.c" (seeappendix E). To allow the routine to be called when WMM receives thecommand above the command number should be a case in the switch statementin the wmm_h ndle routine in this file. Complex data item handlingroutines can be implemented in separate modules, as long they are calledas a result of the switch statement in wmm_handle. Because of the great diversity in data which can be available in a(simulated) robot cell it is impossible to give standard data structuresfor these items. Anyone who wants to extend WMM will have to provide hisown data structures and the routines to put them in a string. At themoment SIG limits the string length to 1000 cha acters so if very largedata structures are to be passed it might be advisable to use files as anintermediary and only transmit file names via the strings. Now another client can request information about the robot from theworld model manager by sending a string with its client number and acommand number. Again we will have to implement a routine in WMM which inreaction to the command retrieves the data stored and puts in a stringwhich can then be send as an answer to the client. For an example of owall this exactly works look at the files "wmm.c" and "fsensor.c"(appendices E and F). To solve the problem

of clients changing the data before otherclients have taken note of it, we have implemented a semaphore-likemechanism. WMM keeps a two dimensional array of dependencies based on theinformation provided by the clients when they make the initial contact.It also keeps an array of flags which indicate whether a client hasnoticed the change or not. A client is not allowed to change a data i embefore all clients which depend on it have taken note of the old value.If it tries to do soit will receive a answer string containing a "WAIT" value, indicatingthat the item has not been changed and that it will have to try again. Ifall clients have been notified WMM allows the data item to be changed andreturn a "OK" value to the client. As an simple example consider one robotwith a force sensor in the gripper. If the robot (simulator) closes the gripperit is not allowed to open it until the sensor (simulator) has taken note that thegripper has been closed. The diagram on the following page (fig. 16) shows us the control flow in



WMM.

Figure 16.

7.4 SEFFAS in practice

We already mentioned the example of the two robots linked by a forcesensor and now we have implemented our World Model Manager it is time toput it all together: In this example SEFFAS will consist of the followingelements: SIG, WMM, 2 BOSROS's and FSENSOR. Whichmeans that we will use two Bosch robots for our example coupled by theforce sensor FSENSOR. Before we make some changes to BOSROS to allow usto run it in 'slave mode' and add the appropriate routines to WMM, wemust implement our force sensor. Implementing a faithful force sensor would mean that in addition tothe direction of movement we would have to take into account the mass ofthe robots, the acceleration of the master robot, dyn mics (compliance)of the force sensor, de-

lay between the master's movement and the slave'sreaction, and perhaps more. But none of these is available in the currentsimulation or even in our real world environment. Again obtaining thisdata and implementing it in the simulation would be a project in itselfso we decided to simplify our problem. When we look at BOSROS we see that our unit of time is defined interms of the number of steps in which to execute a move. The larger thenumber of steps the lon er a move will take. But also the smaller thechange in position in one step. If we set the delay time to one step, addto this a sensor which is compliant enough to allow for the movement totake place and discard all other factors we can base our simulation onthe fact that the force on the sensor will be linear with the size of themovement step and in the direction of the movement. So when we take thedifference between the old position and the new position of the masterrobot we have a vector which gi es us the size and the direction of theforce on the sensor. This is true for movement in the X, Y and Z direction of the robotbut our robot has an additional rotation which changes the orientation ofthe gripper. If we imagine our sensor as a stick held in the grippers ofthe robots changing the orientation will also lead to a force on thesensor (fig.

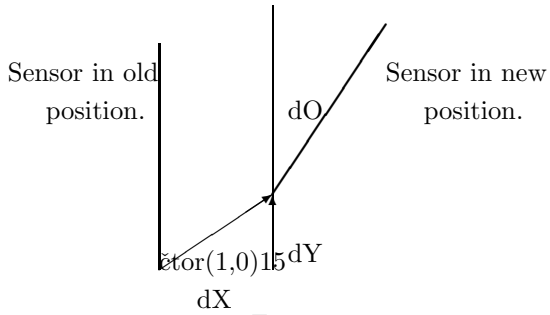


Figure 17.

17). To implement this force we take the end of the stick (in thegripper of the slave robot) as the action point for our force and we alsotake the difference in orientation as a fourth component of our force. Sonow we have a sensor which takes the difference between the old positionand the new position of the end of the imaginary sensor together with thedifference in orientation as a measure of the moment applied to it. To putit in pseudo code:

```
  X_Value = ( New_X - sin ( New_O ) * Sensor_Length ) -            ( Old_X - sin ( Old_O
 Sensor_Length ) -             ( Old_Y - cos ( Old_O ) * Sensor_Length )   Z_Value = New
```

We could put together a function to more faithfully represent thisforce but when we remember that we will only use it to have the slaverobot 'follow the leader', we realize that we can use it directly to havethe slave robot to a new position, resulting in a negation of the forceplaced on the sensor. So our master – sensor – slave interaction will goas follo s: The master does a movement step and sends its new position tothe world model manager. The sensor which polls WMM for

a change in thisposition receives the new position, calculates the difference as shownabove and sends this as a new sensor value to WMM. The slave which in itsturn polls WMM for the sensor value sees the new value and uses it tomove to a new position. This means that in this particular case our worldmodel will consist of nothing more than the position of the robot and thesensor val e which is a result of the movement of that robot. So now we put the necessary functions in WMM and BOSROS. We extendBOSROS with the option to put it in 'slave mode', which means that itwill not take its input from a control panel but instead starts pollingWMM for the sensor value. When there is a force registered by thesensor the slave robot moves to the position indicated by the sensor value.We also addsome code which sends the new position of the robot to WMM when BOSROSruns in master mode. A ess elegant way would have been to add code toWMM to calculate the new position of the robot based on the new linkcontrol values (the angles and translation). This would have saved us theadditional sending of the position. Our FSENSOR program simply consists of a loop which polls WMM for achange in position of the master robot and when it receives one sends thenew sensor value to WMM and shows it graphically in a window. So now weare ready to put this example to work. First we start the SIG-WMMco bination as we used to, by starting mig_server. Next we start twoBOSROSes. We can first start a BOSROS in slave mode and put it in thebackground and then start a BOSROS in master mode in the foreground orstart them in separate windows. It then doesn't matter which we startfirst. We must now align the two robots so we can place our imaginarysensor between them. As we don't have control of the slave robot and oursensor has a fixed length we will have to move the master to (800, -100,193, 0) using the control panel. This will place the master directlyin front of the slave, which is still at its home position. And we cannow put our sensor between themby starting the program fsensor. From now on the slave will try to followthe master (fig. 18). Be careful not to force the slave to go somewhere itcan't because the simulation doesn't incorporate exception handling andthe slave will get hopelessly confused.Figure 18.

8 Conclusions and recommendations

8.1 Conclu ions

When we look back at the goals we had set, namely a simulator environmentcapable of simulating multiple robots and sensors and suitable for interactiveuse, we can draw the following conclusions: Although our experimental set-up of two robots and sensor has littlepractical use, it shows us that we are able to use SEFFAS as a simulationenvironment for multiple robots and sensors. We can easily extend SEFFASfor use with other robot drivers and sensor simulators. Since on a standardSun SPARC w rkstation it takes about a second to perform one step in thesimulation which includes two robots, we can use it interactively withoutunacceptable delays between each step. SEFFAS also shows us that with this simulation we are operation atthe limit of the processing power of simple workstations. Especially ifwe are using multiple robots and sensors the whole simulation will down,depending not only on the complexity of the robots to be visualized but alsoon the sensor complexity, i.e. vision based s nsors like a camera will requirea lot of processing time.The use of a workstation with a graphics accelerator removes the limitimposed by the drawing of the picture, but it introduces a new one in theform of the communication. The communication scheme used, sending stringsover 'stdio' channels, is certainly not the best possible. According to theimplementor of SIG it takes about 30 ms to transmit a string. Thus the useof polling by the clients wastes computing power. While workstations like the S n SPARC's we are using offer thepossibility to provide better and more detailed pictures of the simulatedcell, without additional hardware they lack the power to do sointeractively. The lack of a geometric modelling package limits the extensionpossibilities of SEFFAS. If we want other objects or robots in oursimulation we will need the geometric data in order for SIG to be able toshow them.

8.2 Future developments

As already mentioned the University of Karlsruhe works on the sameproj ct. The SIG simulator is not completely finished so when a newversion become available it is advisable to update our simulator.Furthermore it is advisable to keep an eye on further developments fromKarlsruhe especially the way they plan to incorporate sensors in theirsimulation although the progress of the project is not known. When we look at our own future there are a number of points of thesimulator environment which can be improved. First there is the additionof a programing environment suitab e for programming multiple robots. Thisenvironment must be able to access sensor information and communicatewith multiple programs at the same time. If we are to use SCIL as aprogramming environment for our robot, SCIL would allow us to write ourprograms in C and this is a major advantage. But we don't want to programone robot, we want to be able to pro-

gram a complete robot cell with multiplerobots and sensors. All these robots and sensors are independent runningprograms and SCIL only allows us to a cess functions of the module(s) linkedto it. So in this form it can not be used to control a complete cell. To beable to do this we will need to add our own communication and control routinesto SCIL. Another point of irritation which may stand in the way of futuredevelopments is the limited communication scheme. We would like anothercommunication scheme which allows us to directly exchange informationbetween the programs. One way to implement this would be message passing.In addition to this ther is the general movement to a standardizedwindows environment for UNIX machines called X-Windows. Many programsdeveloped by our university at this moment already use X-Windows. If wewant to port SEFFAS to X-Windows we will have to use another PHIGSlibrary for SIG and we will have to reimplement our control panels, butin addition to conforming to a standard, using X will offer us a new wayto implement our communication. X-Windows supports message passing and wecan use this to implement two way commun cation betweenall the separate programs. It would also make the mig_server programsobsolete. When we look at the future of our robot and sensor simulations therewill be the need to implement more realistic simulators. We might look atobject oriented programming to implement these simulators, which wouldalso nicely complement the object oriented approach of the X-Windowsenvironment. A interesting project would be to implement a force basedsensor in the gripper of a robot which would be able to de œ[7m–More–(78ent design, where the raw sensor data is available in our worldmodel, [Weller] suggests the use of so-called sensor primitives to createabstract data types which cover all types of sensors. In case of morecomplicated sensors like cameras not the image data but the result ofinterpreting the data is available. It may be interesting to examine ifand how these sensor primitives may be used in SEFFAS.

A Installation

To be able to run the simulator program SIG and its clie ts a number ofenvironment variables has to be set. Here follows a list of these variablestheir current value and their function:

`MIG_BIN  /usr/koninga/DiSiSy/bin44`

This is the directory in which the "mig_server" executable is kept.

`SIG_BIN  /usr/koninga/DiSiSy/bin44`

This is the directory in which the "sig" executable is kept.

`PHIGSDIR  /carol/usr/lib/phigs1.1/lib`

This is the directory in which the PHIGS library files are kept.

`DISISY  /usr/koninga/DiSiSyMIG /usr/koninga/DiSiSy`

This is the directory in which the subdirectories "OBJECTS" and"GEOMETRY" are. These two directories contain the data files used by thesimulator. In this "DiSiSy" directory are also all other directoriesrelated to the SIG simulator package.

`MIG_HOST  ‘hostname‘`

This variable has to contain the name of the machine on which the "mig_server' program was started and must be defined on the machine onwhich the client programs are to be run. This is true even if both clients andsimulator run on the same machine. It is advisable to use the predefinedvariable "hostname" to set "MIG_HOST" on log-in.
Finally the source and executables of the client programs, like ARCSand BOSROS are in the directory "/usr/koninga/arcs".

B Graphical data problems

When implementing the Bosch simulation we encountered a problem withth data files used by the SIG simulator. When we examined the data fileswe not only found that one of the files contained two copies of the samedata, but also that the files were of a slightly different format. Thesefiles consist of ASCII data describing the polygons in a picture as follows:

```
  POLYGON  {  4}                   POLYGON     4  -400,-400,0 N 0,0,-1            -400,-4
0,0 N 0,0,-1  -350,-400,0 N 0,0,-1            -350,-400,0 N 0,0,-1   functioning format
```

The remedy is obvious: Place the curly brackets around the number ofvertices in the polygon. The cause of this problem probably lies in theversion of the program used to create the files.

C Changes to SIG

The following changes have been made to the source files of the SIGsimulator package:

```
     sig.c: Added following line to
fix a bug       line 4279: sig_set_light_mode(0);sig_interface.c: Added the following to
      added "wmm.c" and "wmm.o" to the files to be compiled and          linked to "sig"
```

D “WMM.H”

```
file: "wmm.h"-------------------------------------------------------/* This are the comman
T_PSENSOR_VAL  8/* This are answers from WMM */#define WAIT         0#define OK
   0#define SLAVE        256
```

E “WMM.C”

```
file: "wmm.c"-------------------------------------------------------#include <stdio.h>#inc
 the answer message */static int notified[MAXCLIENTS+1]; /* Array with notified flags */
s and sensor values */static POS pos = { 0.0, 0.0, 0.0, 0.0 }; /* Initial unknown robot
h client number is * * associated with a specific client ID, if any.  */{  int i=0;  whi
int client;/* After changing the state of "client" this function resets  * * the notifie
s;    client_id[client]=cid;    for (i=1; i<MAXCLIENTS+1; i++)      depends[client][i]=F
]=cid;      for (i=1; i<MAXCLIENTS+1; i++)        depends[client][i]=FALSE;    }  }  not
nt(cbpp, &client);  if (client==NEW_CLIENT)    sprintf(wmm_ans, "%i", new_client(cbpp));
          /* just a dummy for now */          clear_notified(client);          sprintf(w
     sprintf(wmm_ans, "%i", WAIT);        break;      case GET_SENSOR_VAL:        sprint
 case SET_PSENSOR_VAL:        set_psensor_val(cbpp, client);        break;      case GET
 &pos.y);    get_double(cbpp, &pos.z);    get_double(cbpp, &pos.o);    clear_notified(cl
 client)char **cbpp;int client;/* This function sets a new value of the FSENSOR sensor
nt;/* This function returns the value of the FSENSOR sensor * * to any "client" who expr
```

F “FSENSOR.C”

```
file: "fsensor.c"--------
-----------------------------------------#include <suntool/sunview.h>#include <suntoo
* interpreted as the length of the sensor.                    */#define TRUE  1#defin
opos = { 0.0, 0.0, 0.0, 0.0 }; /* values of the sensor.       */POS oval = { 0.0, 0.0
Pixfont *) NULL, " 100");  pw_vector(Pw,  50,  18,  50, 192, PIX_SRC, 1);  pw_vector(Pw,
arameters are opposite corners of the bar.   */{   pw_vector(Pw, x1, y1, x2, y1, PIX_SRC
, y2, PIX_SRC, 0);   pw_vector(Pw, x1, y2, x1, y1, PIX_SRC, 0);}my_sensor_func()/* Funct
mand);    mig_receive(answer);    sscanf(answer, "%lf %lf %lf %lf",          &npos.x, &
            my_id, SET_FSENSOR_VAL, nval.x, nval.y, nval.z, nval.o);      do      {
ar(150,  70, 150+MIN((int)oval.y,  100),  90);        else          undraw_bar(150+MAX((
       190);        /* Restore the window. */        draw_wind();      }      /* Draw th
al.z,  100), 140);      else        draw_bar(150+MAX((int)nval.z, -100), 120, 150, 140);
%lf\n",           npos.x, npos.y, npos.z, npos.o); */}main(){  Frame frame;  Canvas canv
"WMM: %i %i %i\n",          NEW_CLIENT, FSENSOR_ID, BOSCH_ID);  mig_send(command);  mig_
, 1);  pw_vector(Pw, 250,  18, 250, 192, PIX_SRC, 1);  pw_char(Pw,  20,  30, PIX_SRC, (P
   *   * be called automatically by the SunView runtime handler.                */  noti
```

Literature

[**Gini**] M. Gini
The future of robot programming
Robotica, 5, 235-246
1987

[**Dillmann**] R. Dillmann, B. Hornung, M. Huck
Interactive programming of robots using textual programming
and simulation techniques
Proceedings of 16th ISIR conference
1986 - Brussels, Belgium

[**Huck**] M. Huck
PhD thesis on Robot Programming Systems
1990 - Karlsruhe, Germany

[**Rembold**] U. Rembold, K. Hoermann
Programming of Industrial Robots; Today and in the future
Proceedings of the NATO international Advanced Research Workshop on
Languages for sensor based Control in Robotics
NATO-ASI Series F29
September 1986 - Italy

[**Meijer1**] G.R. Meijer, L.O. Hertzberger
Off-line Programming of Exception Handling Strategies
Robot Control 1988
Preprints of the IFAC-Symposium
October 1988 - Karlsruhe, Germany

[**Meijer2**] G.R. Meijer, T.L. Mai, E. Gaussens, L.O. Hertzberger,
F. Arlabosse
Robot Control with Procedural Expert System
Proceedings of NATO Advanced Study Institute on Expert
Systems and Robotics
NATO-ASI Series F
July 1990 - Corfu, Greece

[**Mul**] Peter Mul
Robot programming and simulation system for robot control

featuring exception handling
December 1988 - Amsterdam, The Netherlands

[**Roth**] Harald Roth
Entwurf und Implementierung eines Visua isierungsmodul fuer ein Robotersimulationssystem unter Verwendung eines Grafikstandards
May 1988 - Karlsruhe, Germany

[**Negretto**] U. Negretto, P. Mul
Task Planning and Simulation of Flexible Assembly Systems
G2-UKA-01.90/1
January 1990 - Karlsruhe, Germany

[**Sun1**] SunPHIGS information brochure
Sun Microsystems Inc.
1988 - Mountain View CA, USA

[**Sun2**] Sun SPARC station 1+ and 1+ GX information brochure
Sun Microsystems Inc.
1990 - Mountain View CA, USA

[**Kate**] Ton K. ten Kate, Richard van Balen, Arnold W.M. Smeulders, Frans C.A. Groen, George A. den Boer
SCILAIM: A multi-level interactive image processing environment
Pattern Recognition Letters 11 (1990) 429441, North-Holland
1990 - Amsterdam, The Netherlands

[**Dondorp**] Er in Dondorp
ASSIM - De Amsterdam Sensor Simulator
June 1990 - Amsterdam, The Netherlands

[**Weller**] G.A. Weller, G.R. Meijer, F.C.A. Groen, L.O. Hertzberger
Sensor based control for autonomous robots
Universiteit Van Amsterdam
Amsterdam, The Netherlands