Experience in Integrating Image Processing Programs

M. Thonnat, S. Moisan, and M. Crubézy

Projet ORION, INRIA – B.P. 93 F-06902 Sophia Antipolis Cedex, France
Monique.Thonnat, Sabine.Moisan, Monica.Crubezy@sophia.inria.fr

Abstract. This paper describes experience we have gained in using program supervision techniques for integration and control of image processing programs. The role of these knowledge-based techniques is to handle high level knowledge about the use of programs together with their effective execution and control. First, the main concepts of program supervision techniques are introduced. Then we present three major classes of application domains where we have applied program supervision techniques: road obstacle detection, medical imaging and astronomy. From these experiments, we conclude that program supervision techniques are good candidates to perform a semantical integration of image processing programs. These techniques are independent of any particular application domain or image processing programs. We propose a general architecture for such systems, which can be slightly tuned for specific purposes. For code designers, semantical integration implies to think from the beginning about how their image processing programs can be integrated and controlled.

1 Introduction

We are interested in knowledge-based techniques for integration and control of image processing programs, called *program supervision*. The role of program supervision is to select programs in an existing library, to run them for particular input data and to eventually control the quality of their results. Various knowledge-based systems have been developed for this purpose in the domain of image processing. We first present our motivations and objectives, then we define more precisely what program supervision is and its importance in vision applications. The heart of the paper describes some prototype applications we have developed for image processing problems. We present three major classes of application domains where we have applied program supervision techniques: road obstacle detection, medical imaging and astronomy. Then, an architecture for program supervision systems and a platform for developing them are presented. We finally conclude that program supervision techniques are adapted to semantical integration of various image processing programs for different applications, at the price of a minimal effort from the beginning of the code design.

2 Motivation

The field of image processing has produced a large number of powerful programs and many different program libraries have been developed. In such libraries the individual programs are integrated from a low-level point of view. But no support is provided to users who need to solve practical image processing problems. Every end-user cannot have a deep understanding of program semantics and syntax. Inexperienced ones may only have a basic understanding of the field of image processing and its terminology. On the other hand, programs implement more and more complex functionalities and their use is equally difficult. If it is too demanding for an end-user to catch the complexity of new programs, these programs will never be widely applied. Our objective is to help users in application fields manage image processing techniques that are needed for their applications. From the analysis of the task of an image processing expert who processes data using a set of programs, we have derived knowledge models related to this task. Based on these models, we propose to support users of image processing libraries, by providing a knowledge-based system for program supervision that takes in charge the management of the library use, freeing the user of doing it manually. This aid can range from an advisory guide up to fully automatic program monitoring systems. The basic idea is to automate the choice and execution of programs from a library to accomplish a user's processing objective. This is done by encapsulating the knowledge of program use in a knowledge base and by emulating the strategy of an expert in the use of the programs. We can summarize our motivation by saying that we are interested in a semantical integration of programs which can also take in charge the control of the quality of their execution.

3 Program Supervision

Program Supervision aims at facilitating the (re)configuration of data processing programs. Such a configuration may involve the chaining of many different programs. It is to note that the program codes usually pre-exist (in a library, for example) and the goal is not to optimize the program themselves, but their *use*. A program supervision knowledge-based system helps a non-specialist user apply programs in different situations as shown in figure 1. It is composed of a program supervision engine and a knowledge base.

The role of the program supervision *engine* is to exploit knowledge about programs in order to produce a plan of programs, that achieves the user's goal. It emulates the strategy of an expert in the use of programs. The final plan that produces satisfactory outputs is usually not straightforward, it often results from several trials and errors. The reasoning engine explores the different possibilities and computes the best one, with respect to expert criteria, available in the knowledge base. The reasoning of a program supervision system consists of different phases, that can be completely or only partly automated: *planning* and *execution* of programs, *evaluation* of the results, and *repair*.

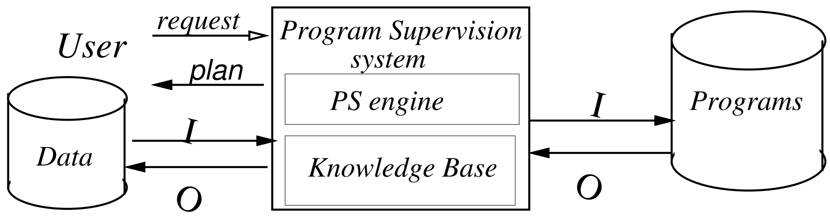

Fig. 1. A knowledge-based program supervision system helps a user (re)use a set of programs for solving a request on input data $\mathcal{I}$ to obtain output data $\mathcal{O}$, as the result of the execution of a plan of programs. It is composed of a program supervision engine and a knowledge base.

The *knowledge base* contains *operators* which are representations of programs (with descriptions of their data and parameters) and of typical combinations of programs, as well as criteria to guide the reasoning process. There are two types of operators: primitive and complex ones. A *primitive operator* represents a particular program and a *complex operator* represents a particular combination of programs. Program combinations correspond to decompositions into more concrete programs, at different levels of abstraction, either by specialization (alternatives) or composition (sequences, parallel, loops, etc.). Moreover, abstract processing functionalities are represented by *goals*. Various criteria, implemented by rules, play an important role during the reasoning, e.g. to choose between different alternatives (*choice criteria*), to tune program execution (*initialization criteria*), to diagnose the quality of the results (*evaluation criteria*) and to repair a bad execution (*adjustment* and *repair criteria*). The contents of the knowledge base should be sufficient for the engine to select the programs, to initialize their parameters, to manage non trivial data-flow and to combine the programs to produce a satisfactory plan of programs, depending on the input data and user's request.

Program supervision is a research domain with an increasing number of work coming from many applicative and technical domains [17]. These research activities are often motivated by a particular application domain (as image processing, signal processing or scientific computing). On the contrary to knowledge-based systems for image interpretation which have been studied for about 20 years, knowledge-based systems for image processing program supervision are more recent. In Japan, numerous teams, belonging both to the research and to the industrial sector, have spent an important effort to this problem (see [18], [14] and [10]). In Europe work has been developed either as research on general tools (as OCAPI [6]) or as industrial tools for a particular application (as the VIDIMUS Esprit project [4]). In the United States early work has been done by Johnston [9] and by Bailey [1]. Recently [8] and [5] have used planning techniques for composing image analysis processes.

4 Prototype Applications

This section shows examples of program supervision systems for image processing in very different application domains.

4.1 Road Obstacle Detection

In the framework of the European Eureka project PROMETHEUS we have done two different experiments using our program supervision techniques for road image analysis.

Stereovision-based Road Obstacle Detection The first example concerns a module performing the detection of objects (such as road, cars, and posts) in road scenes and urban scenes using stereovision data. This module is difficult to use due to the existence of a lot of technical parameters to tune. The data come from two cameras fixed on the top of a moving car. A pyramidal stereovision algorithm based on contour chain points [11] is used to reconstruct the 3D environment in front of the vehicle. Obstacles are detected using 3D as well as 2D information. Integration of the knowledge on the use of this module is needed for robustness, and above all to manage the great variety of the scenes. This integration has been done through a knowledge base, named PROMETHEE. More precisely, a lot of contextual values in the scenes may vary. The luminosity depends on the weather conditions and on the moment in the day, the complexity of the scene depends on the number of objects in front of the car, and the velocity of objects depends on the location of the scene: highway, countryside road or urban street. Due to this variety, it is not possible to fix once and for all the values of the different parameters involved in the processing.

The initial request is the processing of a particular pair of images (taken by two cameras) for object detection. Figure 2 shows an example of such images in the case of an urban scene. First, a 3D reconstruction of the scene is made using a pyramidal stereovision process. Then the 3D reconstruction, together with 2D information (the straight line segments), are used to detect the road, the cars, and the posts. The detected cars are shown in figure 3.

The user needs to take numerous decisions to manage the usage of these programs. A full processing chain contains about 15 different programs to run. Some programs are run several times with different input data; it is the case for instance for the primitive extraction which is made both on left and right images and for different image resolutions. The complete processing contains about 70 program executions. For some steps there are different alternative methods the user can select; for instance there are two different matching techniques each one having several implementations with slight differences. There are a total of 54 values to set to initialize numerical parameters. Among these values several are correlated and there are numerous sensitive parameters.

We have developed a knowledge base to express the knowledge of the usage of these programs. This knowledge base works with the OCAPI [6] engine. Here follow some examples of goals, operators and criteria in this base.

Fig. 2. Stereo images corresponding to an urban scene

Fig. 3. Detected cars in an urban scene

Goals and Operators The knowledge base contains several *goals*, e.g.: *object-detection, pyramidal-stereovision, primitive-extraction, stereo-matching, contour-detection, 2D-3D-cooperation.* For instance, the goal *stereo-matching* corresponding to the actual matching of two images (as part of the pyramidal stereovision process) has 6 input arguments (three pairs of images, with respectively the contour chains, the gradients, and the gradient orientations). The output arguments are an image containing the matched primitives (3D information), and an image containing disparity information. Four choice criteria and one evaluation rule are attached to this goal.

Several primitive and complex *operators* have also been defined in the knowledge base: each of them corresponds to a particular abstract goal. The operator *O-detect-objects* is an example of a complex operator. It corresponds to the goal *object-detection*. It is decomposed into several substeps: first pyramidal stereovision, then a polygonal approximation and finally an object detection using both 2D and 3D information. An example of a primitive operator is the operator *O-meygret-stereo-match* that corresponds to the goal *stereo-matching*. Besides the input and output arguments of its corresponding goal, it also has some additional parameter arguments: two threshold values, respectively on the magnitude and the orientation of the gradient. This operator also contains three initialization criteria and three adjustment criteria.

Criteria The criteria, expressed as production rules, often use information stored in the knowledge base describing the context of use, e.g. information about the physical environment: the type of camera used, the distance from the camera to the front of the car, and whether there is motion or not (motion of the cameras and/or motion of the objects). Some other information concerns the user constraints such as the level of detail that must be extracted from the image (compared to the amount of information in the scene), the desired stereo matching adaptability and quality, or if the processing has to be performed automatically (without possibility of asking a human user to validate the results) or in an interactive way.

- *Choice criteria* contain knowledge on how to select the pertinent operator according to the context of use and to operator characteristics, when several operators are available to solve the same goal. For instance, several operators are available to satisfy the goal *pyramidal-stereovision*. Among them the two operators *O-stereo-pyr1* and *O-stereo-pyr2* perform stereovision using a pyramidal technique and are based on contour chain points. The first operator directly implements this algorithm, while the second one first separates the two interlaced acquisition frames by sampling the lines. The second operator is thus well-adapted when motion is present and when two consecutive lines are not taken consecutively, but at time t and $t + T/2$ (if t is the time, and T is the acquisition period). The rule presented below, attached to the goal *pyramidal-stereovision*, implements such a choice criterion.

if	*context.motion* $==$ *present*
then	use-operator-with-characteristic *with-sampling-for-motion*
comment	"only 1 of the 2 interlaced frames if motion"

- *Evaluation criteria* are most of the time very difficult to express to assess the results of low level goals (like thresholding) since, for example, results are matrices of pixels. But for higher level goals which produce descriptions or object interpretations as results, they are easier to express for experts. For example, the intermediate-level goal *stereo-matching* produces as output matched pairs of primitives which can be counted and compared to a required value. Here follows an automatic evaluation rule of the goal *stereo-matching*:

if *xyz.primitives.number* < *(imchains.primitives.number) / 3*
then assess *number-primitives insufficient*
comment "insufficient matched pairs"

This rule tests the ratio between the number of matched primitives (stored in the data description of the image input data *xyz* of the goal *stereo-matching*) and the number of input primitives (stored in the data description of the argument *imchains*). If this ratio is below the value of 1/3 the rule qualifies the number of matched pairs of primitives as insufficient. Once this problem is diagnosed, it can be repaired later.

– *Initialization criteria* are used to set the parameters of an operator, the value of which depends strongly on the context. For instance, the operator *O-thres-hysteresis* which performs a thresholding by hysteresis (as part of the primitive extraction process) needs the initialization of two input parameters: a high threshold and a low threshold. These thresholds are defined as numerical parameters. The initialization of these values is usually performed in two reasoning phases. In a first phase only symbolical information is used. For example if the user only wants *few* details to be extracted, the threshold should be *high,* as shown in the rule:

  if *context.user-constraints.details == few*
  then *threshold := high*
  comment "few details implies high threshold"

  In a second reasoning phase the symbolical values are translated to concrete numbers, as in the rule:

  if *threshold == average*
  then *heuristic-value := 25*
  comment "average threshold implies 25 as heuristic value"

  This second phase of translation into numbers can be done by selecting a static heuristic value (as shown in the previous rule) or by dynamically computing a value using a calculation method (e.g. probability calculations based on the image histogram).

– When an operator fails because the results are not satisfactory, *adjustment criteria* provide knowledge on how to modify its parameters w.r.t. the type of the failure. For example, we have seen that the evaluation of the goal *stereo-matching* can assess that the number of matched pairs of primitives is insufficient; in such a case, it is possible to adjust the input parameters of the corresponding operator *O-meygret-stereo-match* which are a threshold on the magnitude of the gradient (*thr-m*), and a threshold on the orientation of the gradient (*thr-o*). Below, we show two adjustment rules for this operator. The first one states that the adjustment method for the parameter (*thr-o*) is a technique by percentage:

  if ...
  then adjust-by percentage *thr-o*,
  adjustment-step *thr-o :=0 .3*

  The second rule states that if the number of matched pairs of primitives is insufficient, the two parameters *(thr-m)* and *(thr-o)* have to be increased:

if assessed? *number-primitives insufficient*
then increase *thr-m*,
increase *thr-o*

The PROMETHEE knowledge base [16] contains the representations of 24 programs and 15 combinations of programs. There are 120 criteria among which 64 parameter initializations, 15 choices between operators, 20 result evaluations, and 21 parameters adjustments. Thanks to this knowledge base, all numerical parameters are automatically set. Moreover, all decisions for method selection are fully automated. In the same way all repairs are automatic: they are achieved by adjustment of numerical parameters. The image processing module is general and can be applied to very different images corresponding to various application domains. Two types of problems are automatically detected: a bad primitive extraction and a bad matching. Two other types of thresholding problems can be detected interactively.

Real-time Road obstacle Detection After this positive experiment, we decided to use these techniques to facilitate the integration and control of various image processing programs which have been developed by three external laboratories within the framework of the Eureka European PROMETHEUS project. Thus, a real-time program supervision system, named PLANETE [12], has been embarked on a demonstrator vehicle for traffic safety [13]. It managed the use of a 3-D obstacle detector using telemetry [19], of a mathematical morphology-based obstacle detector [3], and of a motion segmenter [2]. The aim was to provide a system for driving assistance (or copilot) to help the driver in detecting obstacles (cars, pedestrians, etc.) in realistic traffic situations. The demonstration vehicle was equipped with several sensors: 5 CCD cameras, 1 linear stereo camera pair and one telemetry-camera combination, in addition to a number of proprioceptive sensors. The copilot was an intelligent real-time system which processed the information flowing into it from the sensors, used this information to interpret the traffic situation in the vicinity of the vehicle, and communicated its analysis to the driver by means of several interfaces. The total duration of such a cycle was 200ms. In such a real-time real-life application the task of a program supervision system is made more difficult by the stringent requirements of efficiency and speed, and by limited computational (hardware and software) resources. The copilot was composed of a situation interpreter, a data manager, and the real-time program supervision system PLANETE (see figure 4).

The high-level analysis of the traffic scene was done by the situation interpreter, that emitted a request. This request, expressed in terms of zones, obstacles, etc., was then translated by PLANETE into a series of low-level commands—expressed in terms of sensors, program names, data flow, etc.—to the perception programs. The results of the processing done by these programs were then communicated via the data manager to the situation interpreter for further analysis. The role of PLANETE was to make the perceptual processing optimal in the sense of efficiency, timeliness and responsiveness to the rapidly-varying informational requirements of the copilot. PLANETE supervised the running of perception pro-

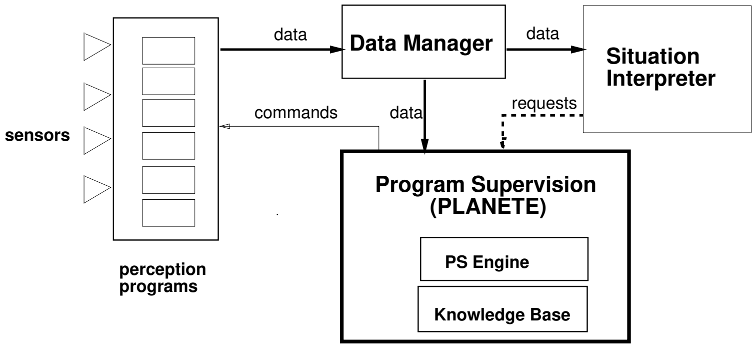

Fig. 4. The copilot architecture.

grams (selection of programs, setting of parameters, etc.) on the one hand, and the efficient management of the computational resources on the other. The prototype vehicle was designed to detect obstacles in different traffic scenarios, such as driving on a city street, crossing an intersection or driving on a highway.

The PLANETE program supervision system succeeded in being fully automated. It was able to manage different types of events: switching from normal driving to overtaking (in that case detection of possible obstacles in the left lane and the status of the blinkers of possible cars in the rear was checked), arrival at a crossing area (a motion detection in both left and right front areas was checked), switching back to normal driving (detection and tracking of obstacles in the front area). It has been demonstrated in October 1994 at the end of the PROMETHEUS project. Unfortunately, the validation of PLANETE has been partial because it has only been tested on several occurrences of a one hour driving scenario in one test circuit.

4.2 Medical Imaging

The third example falls in the medical imaging domain. The objective is to provide clinical users with wider access to evolving medical image processing techniques. In this context, better than fully automating the use of programs, program supervision works in collaboration with users, leaving them with their interpretation (e.g. diagnosis) competence (e.g. for many evaluation criteria). We have developed two applications: Factor Analysis of Medical Image Sequences [7] and Brain Anatomic Segmentation. We detail here our study concerning brain segmentation on 3D MRI images, based on spatial characteristics (e.g. anatomic)[1]. The segmentation process uses computer vision methods, mainly mathematical morphology. First, it binarizes and isolates the informative area (i.e. brain) in images, merely by thresholding methods, then it eliminates the small uninteresting connexities (by means of erosion method) and extracts the

[1] In collaboration with the EPIDAURE team at INRIA, Sophia Antipolis, France.

greatest connected component in the image, assumed to be the brain. Third, lost information is reconstructed thanks to a dilation and an intersection with thresholded image. Last, the segmented brain is presented with superimposition to initial image (see figure 5).

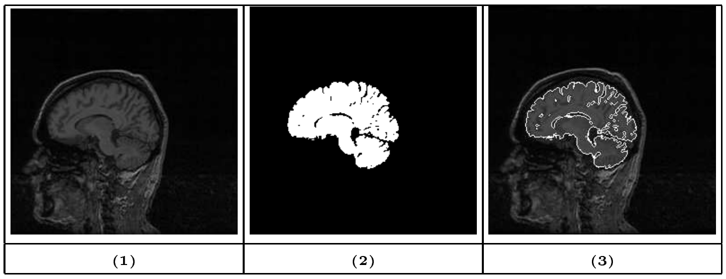

Fig. 5. Example of a segmentation process on 3D MRI brain images: first image is part of initial acquisition, second image shows a slice of the result of segmentation and third image presents a superimposition of the segmented area on initial image, for better visualisation. (Courtesy of Grégoire Malandain, INRIA)

The exact process performed depends on image characteristics, and is solved by its run-time adaptation by the supervision engine. The user needs to take numerous decisions to manage the usage of these programs; a full processing chain contains between 10 and 15 programs to run. There are a total of 27 values to set to initialize numerical parameters. Among these values several are correlated; for instance the size of the structural element for erosion and the one for dilation. The user needs also to take 7 decisions to select between alternative methods.

Medical knowledge important for supervision is not included in programs, but involved in expert's mental process when applying them. We have developed a knowledge base to explicit this process. We have expressed pre-defined high-level combinations of processing steps using symbolic formulation of this knowledge. In particular, medical-related knowledge is stored at an abstract level of the knowledge base, on top of general image processing methodological knowledge. This separation enables the effective supervision of programs and prepares their re-use for other purposes. Another hallmark of this application is that expert's reasoning is highly based on a trial process. Program parameters are difficult to initialize appropriately at once, as they are very sensitive to image quality. The repair phase of program supervision takes here its whole importance, implemented by a mechanism that enables to store the history of problems and transmit them to operators that hold corresponding repair knowledge. Decisions also consider the global objective of the processing as constraints on its realization e.g. to determine the precision with which the brain must be extracted.

The knowledge base contains one goal, 21 primitive operators and 16 complex ones. There are 231 criteria among which 81 parameter initializations, 24 choices between operators, 38 result evaluations, 36 parameters adjustments and 52 repair strategies. Thanks to this knowledge base, 25 values for numerical parameters are automatically set. Moreover, all decisions for method selection are fully automated. For remaining interactive steps an automatic positioning of the 3D image viewer at a pertinent 2D plan is provided to help the user. There are 11 types of problems that are automatically repaired when detected (for instance too high value for binarization threshold, too small brain size, and brain shape irregularity). This knowledge base therefore enables the programs to be used by non-expert users.

However, this program supervision system can be improved. Two remaining numerical parameters must be set interactively (2 binarization thresholds). In the same way, two decisions to automate method selection need information on image descriptions. Currently, this information is obtained interactively. For instance, to choose the right thresholding method information concerning the presence of a gradient of intensity in the image background is needed. Last, all the evaluations are still interactive and their automation requires to develop more image processing programs. For instance, to detect that the brain is too small the automatic computation of the size of the segmented region is necessary. It can then be compared with a reference size.

4.3 Astronomy

The last example is an application in astronomy, where the role of the program supervision system is to automate a complete processing in order to cope with possible variations in the input data (images of galaxies) [15], [20]. The processing aims at analysing astronomical images containing a galaxy. The long term goal is to classify the galaxy contained in the image as astronomical experts do. As astronomical observing systems provide a great amount of high quality data, it is necessary to fully automate their processing.

Galaxy Morphology Description In this case, the image processing objective is to describe the morphology of the galaxy in terms of pertinent numerical parameters. In the current version 45 numerical parameters are computed: 3 global morphological parameters (size, orientation, excentricity) of the galaxy, 2 global photometrical parameters (measuring the errors made by approximating the projected profile with a linear and a $r^{\frac{1}{4}}$ model) and 8 local parameters describing the morphology of 5 different regions of the galaxy (orientation, perimeter, area, coordinates of the center of gravity, length on the major axis and on the minor axis and distance to an elliptical shape). So for each image, the goal of the image processing is the same: detection of the galaxy and computing of these global and local parameters. The programs were already a modular set of 37 image processing programs. The corresponding program supervision knowledge base is a rich one with many of abstraction levels. The criteria are numerous and fully automatic: there are 20 choices between operators, 16 parameter initializations, 11 result evaluations, 14 repair strategies, and 7 parameter adjustments. Thanks to all those criteria, the complete image processing for morphological galaxy description is fully automated and directly provides inputs for an automatic galaxy classification system.

Automatic Galaxy Classification A current requirement in image processing applications is the cooperation between image processing techniques for object description and data interpretation techniques for object recognition. This is the case for the recognition of complex natural objects such as galaxies. Automation of object recognition requires a great amount of knowledge in both image processing and data interpretation. This knowledge corresponds to two different kinds of expertise, this is the reason why we use a distributed approach that separates the two sources of knowledge into two knowledge-based systems (see figure 6).

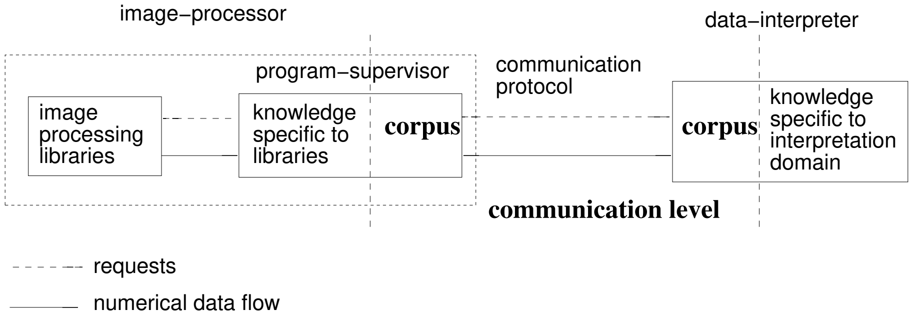

Fig. 6. Two knowledge-based systems (in program supervision and data interpretation) cooperate to automate galaxy classification.

Each system is highly specialized for a task, and they communicate by message sending. The two systems share a common vocabulary, by means of a corpus of typical terms used in vision. The first knowledge-based system is specialized in image processing program supervision. It computes from the image numerical parameters describing the galaxy morphology, as described in previous section. The second one is specialized in data interpretation. It classifies the galaxy described by the numerical parameters. Its engine is based on a taxonomy of galaxy classes and uses a fuzzy mechanism. The complete system provides as output the morphological type plus a detailed symbolic description of the galaxy. The cooperation between these two sub-systems leads to an increasing quality of the results.

5 Architecture of Program Supervision Systems

From these experiments, we can draw a general system architectural framework, that all the previously presented systems conform to. A program supervision system is composed of a reusable program supervision engine, a knowledge base capturing the expert's knowledge about the use of programs, and the library itself as shown in figure 7.

The reasoning of a program supervision *engine* can coarsely be divided into four steps. An initial planning step determines the best (partial) plan to reach the processing goals defined by the user's request. Then the execution of the (partial) plan is triggered, i.e. the individual programs in the plan are executed. The results of the program execution are passed on to an evaluation step that assesses their quality. This evaluation can be performed either automatically by using the expertise in the knowledge base, or interactively by the user. If the assessment on results is negative, it is considered as a failure and a repair step activates the appropriate corrective action. Otherwise the process continues with the planning step for the remaining (sub)goals, if there are any. It is to note that the failure detection and repair mechanisms stay at the level of program supervision expertise. That is to say that a failure is detected only when assessing the *results* of a program execution and repaired only if the expert has been able to provide knowledge on how repairing it. If the failure is caused by the program itself (e.g. incorrect algorithm) or by hardware problems, program supervision will of course be of no help to repair it.

The *knowledge base* encapsulates expertise on programs and processing, i.e. knowledge about the correct use of a library of programs. This primarily includes descriptions of the programs and of their arguments, but also expertise on how to perform automatically different actions, such as initialization of program parameters, assessment of program execution results, etc.

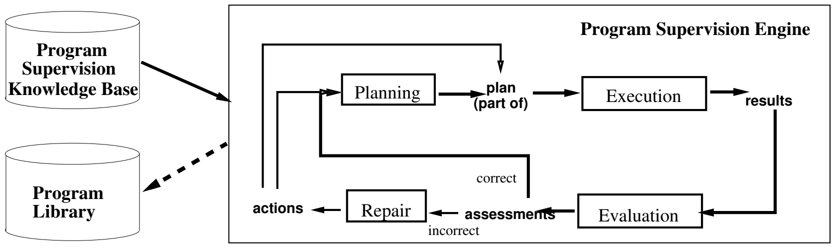

Fig. 7. General program supervision system architecture.

We offer a software platform to develop program supervision knowledge-based systems following this general architecture. This platform, named LAMA, provides knowledge-based system development facilities and different program supervision engines. This generic environment can be used as is, since it is general enough to support the needs of most application domains. As we have stated in the examples, slight differences may exist depending on the particular domain needs. These differences concern both engine capacities and program description details. That is why we offer both a very general set of structures that covers most of the application needs in term of knowledge description and some variants of engines, all based on the same general model (described in figure 7). The platform has already been used to design several systems in program supervision, e.g. most of the examples presented in this paper.

The platform first supplies different engines to run particular reasoning strategies. For example, the planning phase of the engine can be based on either hierarchical-task planning (HTN) techniques, or on operator-based ones, or on a combination of those two techniques. In the same way, the repair phase can be more or less sophisticated. We distinguish two levels of repair: local repair, which means re-execution of the current operator (program) with different parameter values, and global repair, which consists of transmitting a bad assessment to other operators in the plan, until the offending one is found. Local repair may be sufficient when assessment of results is easy to express for almost all the programs of the library. Since problems are detected as soon as their appear they can thus be repaired locally. On the contrary, in other libraries it is possible to assess only few program results. So the detection of failure after the execution of a program often comes from an undetected failure of another. In such cases global repair is necessary to transmit the problem diagnosed and to backtrack in the plan until the faulty program can be re-run. Repair mechanisms are present in all engines, they allow a kind of reactive planning, i.e. the plan can be automatically adapted in reaction to specific problems in the contents of the processing results. This reactivity is part of the knowledge base and the expert should have foreseen the different problems that may occur. Only PLANETE was reactive in the real-time sense to events derived from sensor information.

Concerning the knowledge representation, even if the proposed representation is general and homogeneous, it may be adapted to experts' wishes. Indeed, the experts cannot always express a complete information for every knowledge base piece. Depending on the available knowledge, some parts of the representations may be leaved empty, without affecting the operationalization of the final system. For instance, each program description must not contain all types of criteria. The organization of program combination into complex operators may also be more or less sophisticated. The only important point is to choose an engine able to cope with the available knowledge (with only some loss of performance if there is too few knowledge). The platform provides domain experts with a representation framework to guide them in expressing their knowledge at the expertise level. This framework is implemented by structures representing the main general concepts of program supervision (such as processing goals, operators, data and parameters, rules, etc.) that will constitute the contents of the knowledge base. Experts describe their knowledge bases using a readable language that includes general templates for all these concepts. Knowledge base verification facilities are attached to this language, to help the experts during the design of a new knowledge base for a particular library.

Vision experts hence have at their disposal a generic environment, composed of common tools that can be shared for designing different knowledge-bases systems. Even if a specific domain demands focused adaptation, the platform also allows us to easily tailor an engine to take into account characteristics of target application domains or particular end-users' needs.

6 Conclusion

As we have seen through the examples shown in section4, program supervision techniques can be good candidates –once the low-level program integration has been done– to perform a semantical integration of image processing programs. These techniques are independent of any particular application domain: even if this paper is not exhaustive, we have shown program supervision systems applied to robotics for road scene analysis, to medical imagery, and to astronomy. These techniques are also independent of any particular image processing programs or library of programs. All the program supervision systems presented in the examples were based on the same general model of program supervision for knowledge representation and engine reasoning, that has been derived and adapted with respect to the particular needs of each application.

Semantical integration of image processing programs is needed in two cases: first to develop fully automated systems (as shown in the second example of real-time road obstacle detection and in the last one for automatic galaxy classification), second to develop intelligent help for an end-user (as shown in medical imagery, where the end-user is a clinician).

As a general advice to image processing developers we can suggest from this experience to think from the beginning about how their image processing programs can be integrated and controlled. It can be done by adding to the code information on their functionality, performance, limitations, etc. In particular evaluation and repair knowledge is very important to control the execution of the programs and to adapt the reasoning process. The building of a knowledge base for program supervision is still a big effort for an expert. However, once the knowledge base has been written, program supervision techniques allow code developpers to widely distribute packages of their programs to non-expert end-users.

References

1. D. G. Bailey. Research on Computer-assisted Generation of Image Processing Algorithms. In *Workshop on Computer Vision - Special Hardware and Industrial Applications*, Tokyo, October 1988. IAPR.
2. A. Bellon, J-P. Dérutin, F. Heitz, and Y. Ricquebourg. Real-Time Collision Avoidance at Road Crossing On Board the Prometheus Prolab2 Vehicle. In *Intelligent Vehicles'94 Symposium*, pages 56–61, Paris, (France), October 1994.
3. S. Beucher and M. Bilodeau. Road Segmentation and Obstacle Dectection by a Fast Watershed Transformation. In *Intelligent Vehicles'94 Symposium*, pages 296–301, Paris, (France), October 1994.

4. British-Aerospace. VIDIMUS Esprit Project Annual Report. Technical report, Sowerby Research Centre, Bristol, England, 1991.
5. S. A. Chien and H. B. Mortensen. Automating image processing for scientific data analysis of a large image database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):854–859, August 1996.
6. V. Clément and M. Thonnat. A Knowledge-based Approach to the Integration of Image Processing Procedures. *Computer Vision, Graphics and Image*, 57(2):166–184, March 1993.
7. M. Crubézy, F. Aubry, S. Moisan, V. Chameroy, M. Thonnat, and R. Di Paola. Managing Complex Processing of Medical Image Sequences by Program Supervision Techniques. In *SPIE Medical Imaging'97*, volume 3035, pages 614–625, February 1997.
8. L. Gong and C. A. Kulikowski. Composition of Image Analysis Processes Through Object-Centered Hierarchical Planning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(10), October 1995.
9. M.D. Johnston. An Expert System Approach to Astronomical Data Analysis. In *Proceedings, Goddard Conf. on Space Applications of Artificial Intelligence and Robotics*, pages 1–17, 1987.
10. T. Matsuyama. Expert systems for image processing: Knowledge based composition of image analysis processes. *Computer Vision Graphics Image Processes*, 48:22–49, 1989.
11. A. Meygret, M. Thonnat, and M. Berthod. A pyramidal stereovision algorithm based on cont our chain points. In *European Conference on Computer Vision*, Antibes, April 1990.
12. S. Moisan, C. Shekhar, and M. Thonnat. Real-Time Perception Program supervision for Vehicle Driving Assistance. In Okyay Kaynak, Mehmed Ozkan, Nurdan Bekiroglu, and Ilker Tunay, editors, *ICRAM'95 International Conference on recent Advances in Mechatronics*, pages 173–179, Istanbul, Turkey, August 1995.
13. Groupe PROART. Rapport de fin de contrat de recherche PROMETHEUS-PROART. Technical report, Projet PROMETHEUS, December 1994.
14. H. Sato, Y. Kitamura, and H. Tamura. A Knowledge-based Approach to Vision Algorithm Design for Industrial Parts Feeder. In *Proceedings, IAPR Workshop on Computer Vision, Special hardware and industrial applications*, pages 413–416, Tokyo, 1988.
15. M. Thonnat, V. Clément, and J. C. Ossola. Automatic galaxy description. *Astrophysical Letters and Communication*, 31(1-6):65–72, 1995.
16. M. Thonnat, V. Clement, and J. van den Elst. Supervision of perception tasks for autonomous systems: the OCAPI approach. *Journal of Information Science and Technology*, 3(2):140–163, Jan 1994. Also in Rapport de Recherche 2000, 1993, INRIA Sophia Antipolis.
17. M. Thonnat and S. Moisan. Knowledge-Based Systems for Program Supervision. In *Proceedings of KBUP'95*, pages 3–8, Sophia Antipolis, France, November 1995. INRIA.
18. T. Toriu, H. Iwase, and M Yoshida. An Expert System for Image Processing. *FUJITSU Sci.Tech.Journal*, 23.2:111–118, 1987.
19. L. Trassoudaine, J. Alizon, F. Collange, and J. Gallice. Visual Tracking by a Multisensorial Approach. In *First IFAC International Workshop on Intelligent Autonomous Vehicles*, pages 111–116, Southampton, (UK), April 1993.
20. R. Vincent, M. Thonnat, and J.C. Ossola. Program supervision for automatic galaxy classification. In *Proc. of the International Conference on Imaging Science, Systems, and Technology CISST'97*, Las Vegas, USA, June 1997.