Integration of image processing procedures : OCAPI, a knowledge-based approach

Véronique Clément, Monique Thonnat

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1307

Programme 6
Robotique, Image et Vision

INTEGRATION OF IMAGE PROCESSING PROCEDURES : OCAPI, A KNOWLEDGE-BASED APPROACH

Véronique CLEMENT
Monique THONNAT

Octobre 1990

RR-1307

Integration of image processing procedures:

OCAPI, a knowledge-based approach

Véronique Clément and Monique Thonnat
INRIA Sophia Antipolis
2004, Route des Lucioles
F-06565 Valbonne Cedex, France
Tel: (33) 93-65-78-57 Fax: (33) 93-65-77-66
e-mail: vclement@mirsa.inria.fr thonnat@mirsa.inria.fr

Abstract: This paper deals with the integration of image processing procedures. Three kinds of integration are distinguished: physical, syntactical, and semantic integration. The notion of semantic integration of programs is developed here; in this kind of integration, the function of the programs, and how to optimize their utilization are explicited. After a short presentation of several recently developed systems performing semantic integration of programs, we describe how we model knowledge and reasoning in our system named OCAPI. Two examples of use of OCAPI are shown: the first example is the integration of a stereovisual process, the second one is the integration of a system for the morphological description of galaxies. Finally, the model and the mechanisms used in OCAPI are compared to those of the previously presented systems.

Keywords: image processing, expert systems, integration of programs, control of execution, stereovision, astronomy.

Intégration de procédures de traitement d'images :

OCAPI, une approche à base de connaissance

Résumé : Ce papier traite de l'intégration de procédures de traitement d'images. Trois sortes d'intégration sont distinguées : l'intégration physique, syntaxique et sémantique. La notion d'intégration sémantique de programmes est développée ici : pour ce type d'intégration, la fonction des programmes ainsi que la manière d'optimiser leur utilisation sont explicitées. Après une brève présentation de plusieurs systèmes récemment développés pour l'intégration de programmes, nous décrivons comment la connaissance et le raisonnement sont modélisés dans notre système, nommé OCAPI. Deux exemples d'utilisation d'OCAPI sont montrés : le premier est l'intégration d'un processus de stéréovision, le second concerne un système de description morphologique de galaxies. Enfin, le modèle et les mécanismes utilisées dans OCAPI sont comparés à ceux mis en œuvre dans les autres systèmes présentés.

Mots clés : traitement d'images, systèmes experts, intégration de programmes, pilotage d'algorithmes, contrôle d'exécution, stéréovision, astronomie.

1 Introduction

The integration of a set of existing programs is a very important and difficult task. The major difficulty of the integration phase is that most of the time modules are developed independently, either by different specialists or at different periods of time. Knowledge of use of these modules is not included in the code, but split into several brains or forgotten.

Although this domain is still young, three types of integration (physical, syntactical, and semantic) can be distinguished. More precisely:

- usually integration of programs is understood as the adaptation of the algorithms to a specialized hardware; this kind of integration is what we call **physical integration** of software onto a given architecture;
- integration of a set of programs in a command language or an interface system is what we call **syntactical integration**; it is the integration of the syntactical information needed to run the programs;
- finally a third kind of integration is possible: the integration in a system of knowledge of the goals of the programs, knowledge of conditions under which they are applicable, and knowledge of relations between the programs; we speak of **semantic integration** or intelligent integrated systems.

Usually, image processing systems are composed of a set of programs or subroutines which can be used directly or through a command language. These systems only provide a physical integration of the procedures or at best a syntactical integration; that is the use of abstract commands instead of operating system or programming language syntax. If the set of programs are semantically integrated, we may have an intelligent image processing system with a certain degree of autonomy. Such intelligent image processing systems could facilitate the developing of new image processing applications since certain tasks can be automated (like the selection of the best programs to reach a goal or the initialization of some parameters); moreover, if their knowledge is sufficiently developed, this can extend the use of image processing systems: they can be directly used by non specialists in image processing (like biologists, astronomers...) or integrated on autonomous systems working in complex and variable environments (like mobile robots or flexible arms). Physical, and syntactical integration of programs for robotics purposes only deals with the problem of **robustness** of the processing (the processing is fixed, but the input data can be slightly noisy); with semantic integration of programs it is also possible to deal with the problem of **flexibility** of the processing (the processing can adapt itself to various conditions).

Various methodologies, generally based on artificial intelligence techniques, have been investigated to develop intelligent image processing systems. So a clear distinction has

to be made to discriminate between those which are intelligent algorithms, intelligent interpretation systems, or intelligent integrated systems. A classification can easily be performed according to the nature of the knowledge these systems contain **explicitly**:

- **intelligent algorithms** express in detail *how* programs work, and describe explicitly the *internal mechanisms* of the algorithms. This kind of knowledge is interesting for innovation, generalization or educational purpose. One can mention as examples of such intelligent algorithms some interesting work in segmentation (the system developed by Nazif and Levine [Naz 84], LLVE [Mat 89], and VISIONS [Han 87]).

- **intelligent interpretation systems** contain explicit knowledge on the *modelling of objects* of the world. As examples of such intelligent interpretation systems, one can mention ACRONYM [Bro 81] which uses a geometrical modelling of domain objects, CLASSIC [Tho 88] which uses a semantic modelling of classes, and the system developed by Ikeuchi [Ike 88] which generates an object recognition program from a 3D CAD model of the object.

- **intelligent integrated systems** contain (or have to contain) all the knowledge needed for the selection and the use of programs seen as black boxes. Some algorithms in the library can of course perform object recognition, and they can even be intelligent; but they are considered by the integration system like a black box. In such intelligent integrated systems, *what* programs do (their goal) and *when* (under which conditions) are expressed explicitly.

In this paper, we will focus on semantic integration of programs in image processing. Work in this domain is very recent [Joh 87] [Tor 87] [Tan 88] [Sat 88] [Bai 88b] [Cle 89]. So, in the next section, we present some systems which are representative ones, either for their state of development or their mechanisms. In section 3 we present the model of reasoning and knowledge used in our system named OCAPI. Then in section 4 we describe two examples of integration of image processing based on OCAPI: one for stereovision, and the other for morphological description of galaxies. In the last section, we compare the models used in the systems we have presented.

2 Recent work in semantic integration

Integration of image processing modules has been recently addressed using intelligent systems. In this section, five such systems are presented: the goals of the authors are quite different: interactive help versus fully automatic system, domain specific tool versus general system, generation of programs or scripts versus execution of commands.

2.1 Expert assistant

In the field of astronomy, Johnston introduces the concept of an expert *Data Analysis Assistant*. In fact, astronomers manipulate image analysis systems which offer numerous functionalities. Depending on how and when the data have been obtained, astronomers often have to use various image analysis systems. These systems are usually implemented as command languages. So, Johnston has proposed Expert assistant [Joh 87], a system of generation of commands, independent from the image analysis system. A prototype has been developed using the commercial expert systems shell KEE [Int 85]; it addresses the problem of CCD calibration as a simple test case for its methodology. Two different image analysis systems are available (MIDAS and SDAS/IRAF).

The expert system reasons on data analysis operations, and on a simple symbolic description of the input data; it does not access the data. In the first step, an optimized sequence of commands is generated. In the second step, the generated sequence is executed by the image analysis system.

So this expert system simply generates optimized sequences of commands given a simple symbolic description of the data.

2.2 Toriu's system

Toriu *et al.* have been interested in the general domain of image processing, without targeting any specific domain of application. They are concerned with improving the productivity of beginners and non specialists who have to develop specific algorithms for an application. So, an expert system [Tor 87], working on the FIVIS image processing system, has been implemented in Lisp. This system provides functions that would help users to construct their application, by automatically selecting and combining various image processing modules. It is not just a prototype, but already a commercial tool: 120 image processing modules are available, and 40 image attribute names are used.

The user chooses a goal among a predefined list of goals, such as segmentation, image quality improvement or feature extraction, and expresses characteristics of the image to be processed (image type, noise level, texture...). Inferences are done to choose the processing; then for each step of the processing, a module is selected, and executed by the image processing system. So, selection and execution of the modules are done in the same process, and are interleaved (a module is selected and immediately executed).

This system offers a conversational help to the user to perform image processing.

2.3 EXPLAIN

The work presented in [Tan 88] is also involved in the general domain of image processing, without any specific application field. Tanaka and Sueda developed a system which assists the non-expert in using a package of image processing algorithms to obtain a required image from a given one. As special feature, a knowledge acquisition facility is incorporated; it captures knowledge about image processing procedures through interaction with the domain expert. EXPLAIN has subsystems for consultation, and image processing. They are implemented as a set of PROLOG clauses.

The consultation subsystem interacts with the user to guide image processing, and suggests an appropriate course of actions. The problem is specified using keywords like enhancement or segmentation, and some approximated properties of the image to be processed (color/black and white, noise level, contrast). The process decomposes the given requirement into a sequence of image processing algorithms. Then these algorithms are run, and their results are displayed to the user; in case of unsatisfactory results, the values of the parameters are modified, or another command is tried.

EXPLAIN is an image processing expert system; it runs the commands on the images, and provides a trial-and-error mechanism to deliver satisfactory results.

2.4 DIA-ES

H.Tamura and K.Sakaue [Tam 84] proposed the DIA-ES system (Digital Image Analysis - Expert System). It allows, in a semi automatic way, the design of an algorithm for a specific application. It has been used to design algorithms for a problem of pattern recognition [Sat 88]. DIA-ES is composed of two subsystems, the semantics and the syntactic processor, which are developed on a symbolics lisp machine (using Uranus prolog/KR).

Given an image and a problem, the semantics processor selects the appropriate processing modules. This is done using knowledge about image processing methods, and interactively with the application designer. The syntactic processor has been presented by K.Sakaue and H.Tamura in [Sak 85]: given a sequence of abstract commands, it generates a program using knowledge of the data, and the algorithm structures of the selected modules; each one is described with its name, and informations concerning its arguments. An application has been developed with the library SPIDER [Tam 83].

So, from an image, its characteristics, and a goal, DIA-ES generates sequences of abstract commands (semantics processor), and optimized programs (syntactic processor).

2.5 D.G.Bailey's system

D.G.Bailey is interested in making easier the development of image processing algorithms for people who are specialists in some application domain but not in image processing. A prototype presented in [Bai 88b] has been developed, but not yet used for some specific application. The system is composed of the algorithm generation subsystem implemented in PROLOG, and of an image processing subsystem which is currently a command-based system very similar to VIPS [Bai 88a]. But the design has been done in such a way that any other image processing could also be used.

In the first step, the system interacts with the user to determine precisely the type of image processing task to perform; then recommendations are made on the way to obtain a representative image to develop the algorithm on. In the second step, an architecture of algorithm is selected among the architectures described in the knowledge base, according to the type of task to perform; this is done using selection rules. Each architecture contains a decomposition of a main task into substeps. For each substep, an operation which can be performed by the image processing subsystem is selected, and performed. The results are evaluated by the user; if they are not satisfactory, another operation or sequence of operations is tried. When all the results are satisfactory, the algorithm is provided to the user, and can be tested on other images. So, this prototype, which is under development, provides computer assisted generation of image processing algorithms.

2.6 Conclusion

We can notice that all the presented systems, have the architecture of an expert system; they are expert systems themselves or generator of expert systems, depending on their generality. In fact, this methodology allows the separation between particular knowledge about an application domain and programs, and the control of the reasoning for integration; so the development and the reutilization of such systems is facilitated.

3 The OCAPI system

3.1 Objective

In order to perform semantic integration of programs, we are concerned with the problems of modeling knowledge and reasoning which are used in the development of image processing systems. Due to the necessity for generality, the model has to be independent from any specific application or domain of application, and even from any image processing system. First, we propose a model verifying these requirements; second, we describe OCAPI, an implementation of this model which automates execution of image processing;

i.e., given data to be processed, a goal to reach, and eventual constraints on the results, the system generates, performs, and optimizes the processing. Systems mentioned in the previous section do not take into account the same constraints. For instance, Expert assistant cannot run the script of commands that it has generated; Toriu's system is heavily depending on the FIVIS image processing system; and none of those systems allows a fully automatic mode of utilization.

In this section, our model and its implementation are presented. First, a model of the reasoning performed in semantic integration is described (the types of reasoning, and the various techniques). Second, a model of knowledge involved in semantic integration of programs is presented (typology of various concepts, and relations between them). Then, we conclude on the implementation of the OCAPI system. Some examples of integration using OCAPI are given in the following section.

3.2 Model of reasoning

[Mat 89] and [Vog 86] discuss how to reason during the development of an image processing application. This can be summarized using examples as follows:

-1- selection of substeps or subgoals: for example: to reach a certain goal, one has first to perform a segmentation, then to select the biggest region, then to describe its shape;

-2- selection of operators, algorithms or methods: for example: for image segmentation three methods can exist while, in a particular context, we choose the operator seg1;

-3- initialization of the input arguments; for example: set threshold t=0.45 and input image=im21;

-4- effective execution of programs ; for example: run the command " seg1 im21 -t 0.45"

-5- test of the results (satisfactory or not); for example: if the size is below 30, the segmentation is too fine;

-6- new execution of programs with other values of parameters; for example: if the segmentation is too fine, increase t by 10%;

-7- if the results are still not correct, selection of other methods; for example: if the operator seg1 fails then use the operator seg2.

Phases 1 and 2 correspond to a **planning** reasoning. In fact, planning is "to describe a set of actions (or a plan) that can be expected to allow the system to reach a desired goal" [Hen 90]. Here, the actions are the programs; a plan is a chaining of programs,

while goals are functionalities to provide. Phases 3 to 6 execute the generated plan, and control its quality; so these phases correspond to a **control of execution** reasoning. The last phase corresponds to a **replanning** reasoning; i.e., execution of the plan is stopped, and the plan has to be changed; this necessitates a new planning phase.

Two kinds of reasoning need to be developed: planning, and control of execution.

3.2.1 Planning for integration

In this paragraph, the question is: which planning technique is adequate for semantic integration of programs?

In problem solving, planning is to find a plan for achieving a goal according to a given state of the world. A plan is defined as an ordered list of actions to perform. Three main techniques have been developed in planning: non hierarchical, hierarchical and skeletal planning [Bar 82]. While the two first methods concentrate on the reduction of conflicts in the plan steps, the last one emphasizes the expertise of the specific domain. The third method, also called script-based planning, assumes that abstract or skeletal plans exist. A skeletal plan contains the basic steps to solve a problem. First, the best skeleton is selected, then refined.

In image processing, we have few sensitive conflicting subgoals; so the two first techniques are not necessary. As we hold knowledge, at several levels of abstraction, of typical processing methods, a skeletal technique providing a refinement mechanism is appropriate. Another point is the handling of real world data; we have to deal with unreliable or uncomplete information; the state of the world (initial, final or even intermediate) can not be described exhaustively. So we have to use a technique based on the description of the actions to be performed, and not on the different possible states of the world. Given these prerequisites, a skeletal planning technique proves to be adequate for the planning phase. A plan can eventually be modified during a replanning phase by replacements or insertions of actions.

3.2.2 Control of execution for integration

We are interested, here, in defining the type of control of execution which is the best adapted for the integration of programs.

Four main types of control of execution [Hen 90] have been developed:

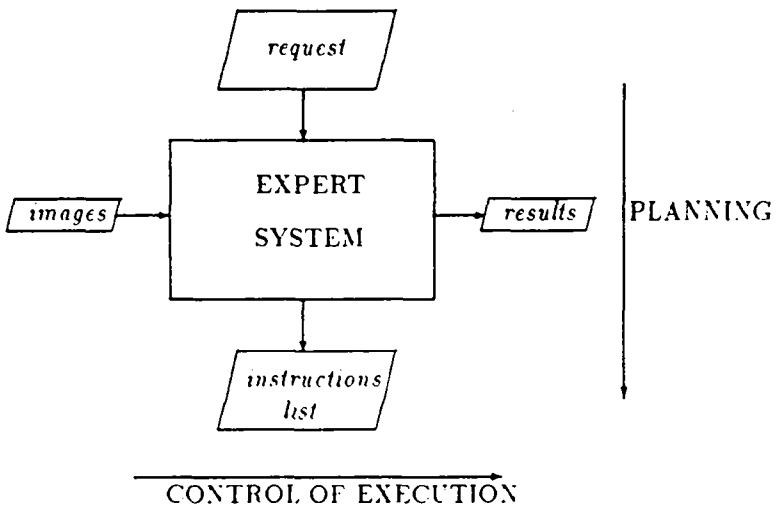
- prediction of failures, and planning of additional actions which wait until the world corresponds to what is expected (example: waiting for a light to turn green before crossing a street);

- prediction of failures, and integration in the plan of tests and correcting actions;
- prediction of failures, and integration in the plan of tests and alternatives;
- integration of the planning process, and control of the execution process.

In our context, in image processing, we do not handle secure actions: predicting exactly the behaviour of an image processing operator on an image is not possible. So, a trial-and-error strategy is needed, as well as the interleaving of planning and control of execution processes.

3.2.3 Reasoning for integration

In conclusion, two main tasks have to be performed (Figure 1).



- Figure 1 - Reasoning for integration

First, a planning phase reasons about an appropriate plan to guide the processing of the given images. This step determines the list of instructions to run according to the request of the user: this is done using a progressive refinement strategy. Second, the control of execution of the plan is done through trial-and-error experiments. The quality of the results obtained from the input images is checked (evaluation of the results), and in case of unsatisfactory results, a mechanism adapts the values of the parameters of the image processing algorithms (adjustment of the parameters). The sequence of operators can also be modified using a replanning mechanism if needed.

3.3 Model of knowledge

In this section, we describe in detail the model of knowledge we have defined for semantic integration of programs. First we define a typology of the different kinds of knowledge we need, then we describe how all these concepts are related and structured.

3.3.1 Typology of the various concepts

In order to model knowledge of integration, and to facilitate the building of knowledge bases, the important concepts have to be found.

First of all, when we process an image, we always have in mind an objective or a goal to reach. This notion of goal is close to that of task we need to serve. In the following section, we use the term *goal* which is defined below:

> a **goal** represents an image processing functionality. This functionality or objective can be reached by a program or a complex processing. For instance, *to smooth an image* is a goal; in the same way, *to count objects* in an image is another one.

We can note that programs in a library implement image processing functionalities or tasks to perform, but they **are not** image processing functionalities. In fact, we distinguish this abstract notion of *goal* from that of available *operators:*

> **operators** are actions which can be performed. They contain specific knowledge to solve a given goal. An operator can be a particular program (it will be referred to as **primitive operator**) or a particular sequence of processings (it will be referred to as **complex operator**). For instance, the primitive operator *median3* is a program implementing a median filter with window 3×3, while the complex operator *count-objects1* is a succession of several processings (smoothing, segmentation, sorting, enumeration).

In addition to these concepts, two other notions must be introduced: notion of context and that of request. First, in order to find the best sequences of programs, the *context* has to be taken into account:

> the **context** is a description of the input data, their conditions of acquisition, the application domain, and even semantic information on the supposed contents of the scene. For instance, a context can be: *c-astro: the application domain is astronomy, the optical instrument is a high quality telescope, the digitizer is a microdensitometer, the scenes are extended sources scenes, and stars are present.* Another example would be *c-robot-in: the application domain is robotics, the optical instrument is a CCD camera, the digitizer is a standard one, scenes are indoor scenes.*

Second, a problem in vision is expressed by its goal (or functionality), but also by the data to be processed, and eventual constraints on the results to be obtained. The term of **request** will denote all these kinds of information:

a **request** states which goal has to be reached, the data of the particular case to work on (for example, the name of the input image), the required quality for the results, and the context. A request is thus a query to solve a goal for a specific case. For instance, a request can be *r-cont: contours detection of the image "view1"*; another one can be *r-region: region segmentation of the image "view2", the size of the regions must be greater than 2500 pixels, and the context is "robot-in"*.

Operators (primitive or complex), and goals work on a list of input and output arguments as programs do. For instance, the goal *smoothing* has two arguments: one input argument (the input image), and one output argument (the smoothed image); the primitive operator *median1* implements a way to reach the goal *smoothing*, and has three arguments: two input arguments (the input image, and the window size), and one output argument (the smoothed image). We can distinguish two classes among the arguments, *data* and *parameters:*

data arguments have fixed values, which are set (input data) or computed (results). In the previous example, *input image* and *smoothed image* were data arguments. Often, operators performing the same goal have the same data arguments;

numerous image processing operators have tunable parameters. They will be referred to as **parameters arguments**; they have adjustable values and are always input arguments. In the previous example, *window size* was a parameter argument. The description of these arguments is important in the knowledge base; in order to optimize the processing, we not only need to know their value (e.g.: *window size = 3*) but also their range (*minimum = 3, maximum = 11*), an eventual default value (*default = 3*), and their type (*integer* or *real*). These arguments, and their characteristics are specific to each operator.

We have seen in the previous section concerning the model of reasoning that we have several phases in integration: how to choose among methods, how to initialize input arguments, how to execute programs, how to evaluate results, and eventually how to modify the processing with the determination of new input values for programs or selection of other programs. From this description, we can distinguish four notions: choice of operators, evaluation of results, initialization of parameters, and adjustment of parameters:

by the notion of **choice**, we mean the knowledge of how to select, among all the available operators, the operator(s) which is (are) the most pertinent, according to the data, and the context. For example: *if the images have a high contrast, use a contour based segmentation operator.*

by the notion of **evaluation**, we mean the knowledge of how to assess the results provided by the selected operator after its execution, taking into account constraints expressed in the request. For example: *if the number of the segmented regions is greater than 100, the segmentation is too fine.*

by the notion of **initialization**, we mean the knowledge of how to initialize values of input arguments. Various mechanisms are required, as default values or computation method depending on the values of other arguments. For example: *for the parameter window size, the default value is 3 .*

by the notion of **adjustment**, we mean the knowledge of how to modify values of parameters after a negative evaluation. For example: *if the segmentation is too fine, increase the value of the threshold th-sigma.*

3.3.2 Relationship between these concepts

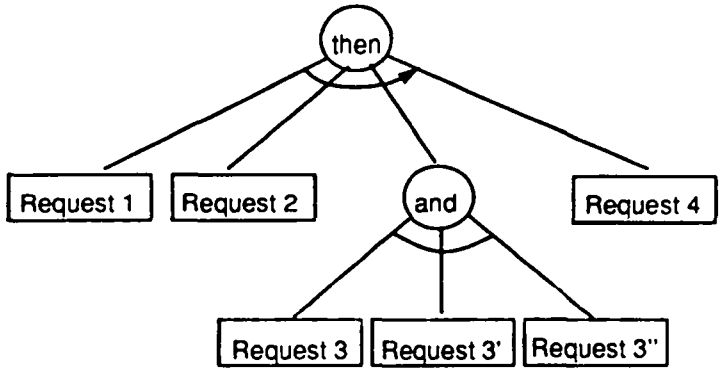
In the previous paragraph, we have defined several concepts (goal, primitive or complex operator, context, request, data or parameter arguments, choice, evaluation, initialization and adjustment). These notions are not isolated, but are related together.

Relationship between goals, operators and requests: Operators are always related to a precise goal; but the level of abstraction of a goal (abstract notion), and of its related operators (concrete executable actions) are different. A complex operator implements a set of processings, which is referred to as a **decomposition**. A decomposition is not a set of programs, neither a set of goals, but is a set of requests. To be more precise: a decomposition is not just a set of programs, because, for example, we can mention the case of a particular operator for stereovision which can be decomposed into two steps: extraction of primitives, and matching of primitives; but if several operators (complex or primitive) are able to perform extraction of primitives, such a decomposition does not impose which one to use. In the same way, a decomposition is not just a set of goals; in fact, in a particular decomposition, data flow of input and output arguments have to be managed: for example, the output of extraction of primitives is one input to the matching. To dictate some constraints about the results of each step must also be possible: matching of primitives will be performed only if the number of extracted primitives is significant for the given problem. So, each step of a decomposition is a request to solve a goal in a special context, with restrictions on the input, and eventual specifications on the results.

Relationship between the steps of a decomposition: The set of requests involved in a decomposition can be seen as a tree where the leaves are the requests, and the full nodes express the temporal link between the requests.

Two types of temporal links are used, representing:

- *sequentiality:* Request 1 THEN Request 2.
  For example: filtering THEN contour detection THEN chaining.
- *parallelism:* Request 1 AND Request 2.
  For example, in stereovision: extraction of primitives on the left image AND extraction of primitives on the right image.



- Figure 2 - An example of a tree of requests

The example of Figure 2 corresponds to requests which are related on this way: first Request 1 then Request 2 then [Request 3 and Request 3' and Request 3"] then Request 4.

Relationship between choice, evaluation and goals: We have seen that the available operators to perform a goal are related to it. But, the goal has to know how to select among them, the operator which is the most suitable for a specific problem: the knowledge of how to choose the operator is global to the goal, and not local to each operator. Evaluation of the results is also a mechanism which has to be related to the goal. In fact, the goal knows what results are required: it chooses the way to perform them, and then has to verify their conformity to the requirements. The criteria of evaluation must be common for the same goal, because the evaluation of the results has to be independent from the operator which performs the processing. So a goal can be seen as an item able to decide how to solve a given problem, and to validate the results.

Relationship between initialization, adjustment and operators: An operator has to know how to initialize its parameters. In fact, each operator uses its input parameters in a specific way (meaning, format, range, sensitivity...). *A fortiori*, knowledge about adjustment of the input parameters depends on the operator: for a given bad result, the sensivity of the various parameters depends highly on the particular algorithm and implementation for a program, and on the particular decomposition into subrequests for a complex operator. So these mechanisms must be located on each operator. An operator is an item able to manage its parameters.

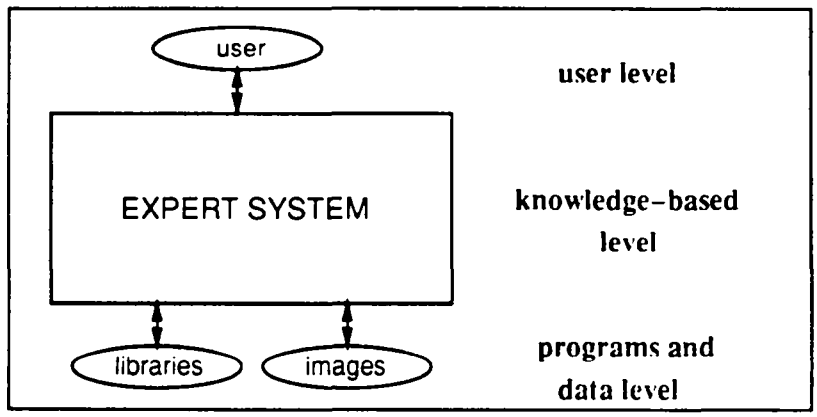
3.3.3 Knowledge for integration

We can summarize the model as follows: the user submits a request which relates to a goal, a set of input data, and a specific context. As a goal can be solved by several operators, choice knowledge is used to select an operator among them. An operator can be a program (primitive operator), or can have a proper decomposition into subrequests (complex

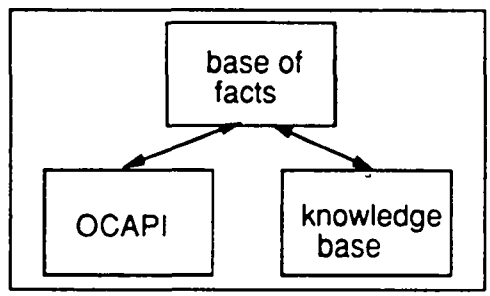
operator). Before the execution of the operator, initialization knowledge sets the values of its parameters. The results of the execution are estimated by evaluation knowledge. Adjustment knowledge modifies the values of the parameters in case of unsatisfactory results.

3.4 Implementation

This model of reasoning and knowledge for semantic integration of programs has been implemented within an expert system generator architecture. Figure 3 explains the relations between the user, the knowledge base level, the data, and the programs. Figure 4 details the architecture of an expert system built with OCAPI; it is compound with a knowledge base, a base of facts, and the control structures (OCAPI).



- Figure 3 - Use of an expert system built with OCAPI



- Figure 4 - Internal architecture of an expert system built with OCAPI

OCAPI has been developed in a lisp dialect: Le_Lisp, [1] and its object oriented facilities.

[1] Le_Lisp is a registered trademark of INRIA

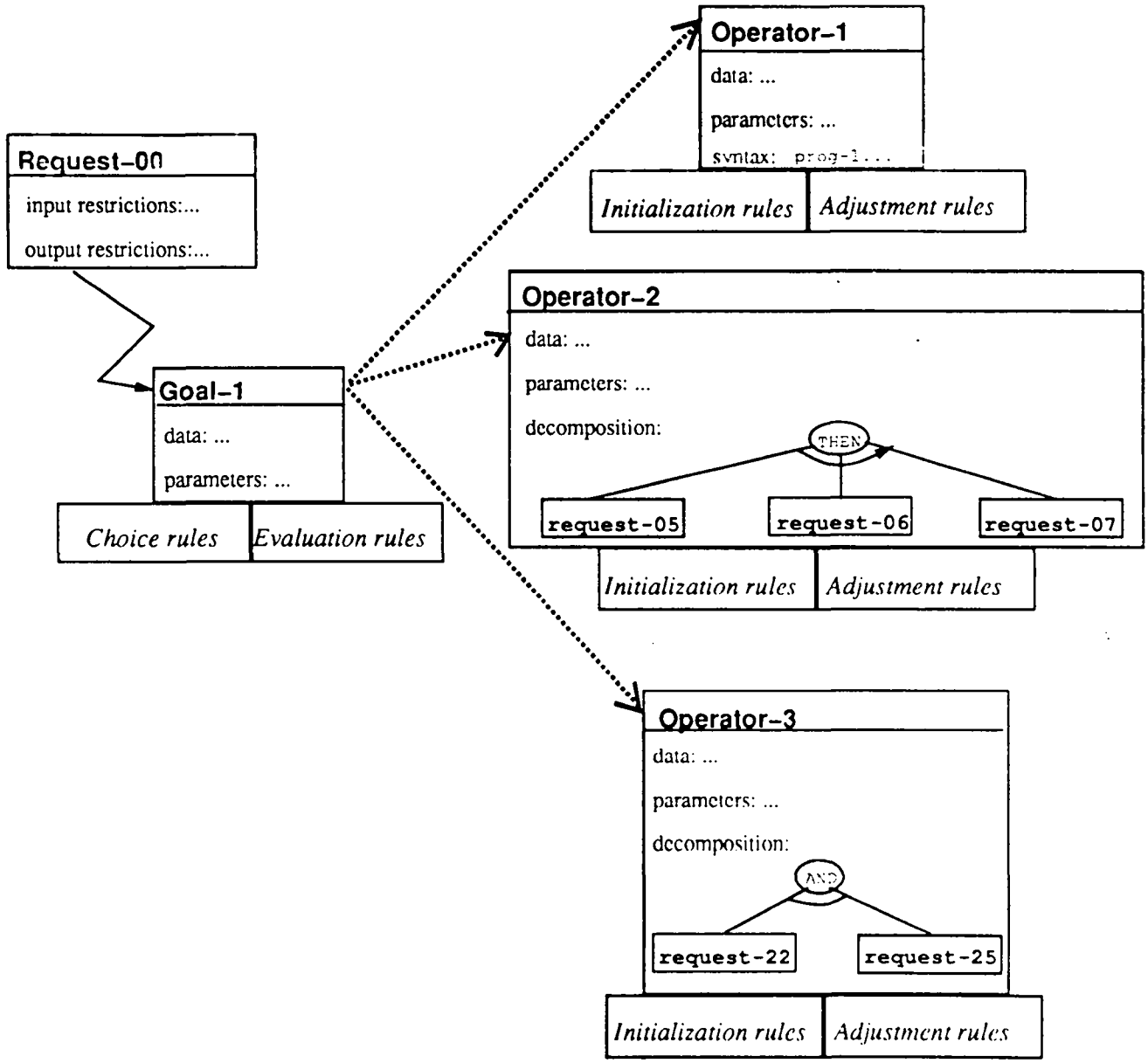
3.4.1 Implementation of the knowledge model

We have seen that strong relations exist between the various concepts we have introduced. These relations structure the knowledge base in order to facilitate the expression of the knowledge and also its utilization. The static or descriptive knowledge is implemented with frames (or structured objects), i.e. goals, operators, context, requests, and arguments. For heuristic criteria concerning choice, evaluation, initialization, and adjustment, production rules are used.

Objects: Objects are a very interesting way of structuring the knowledge base; grouping in the same object (so in the same location) all the information about an algorithm or a method, is efficient (no inferences to execute), clear (readable), easy to access, and secure (a local modification will not have unwanted consequences as it happens when all the knowledge is expressed with rules). Various types of objects (goals, primitive or complex operators) structure the knowledge base into several levels of abstraction. So, incremental development of large knowledge bases by several specialists becomes possible. For instance, knowledge about chaining can be expressed by a specialist of this technique; then an expert on stereovision can build a knowledge base which, if needed, refers to the goal *chaining* : so, the goal *stereovision* will be partly solved using the knowledge base developed by the specialist in chaining.

Rule bases: The knowledge base is also structured using several small bases of production rules. Rules are grouped depending on their semantics (choice, evaluation, initialization and adjustment rules): in addition, these rules, which are typed, are attached to the only objects they are concerned with. Structuring the base in this way greatly improves the efficiency (only few rules are scanned at a given moment), and facilitates the development of knowledge bases by several experts (the same vocabulary can be used for different items: the same words *threshold* or *input image* can be used for the parameters of several distinct goals).

Figure 5 shows the **relations between the objects, and the rule bases.** A request (for instance *Request-00*) is related to a particular goal (here *Goal-1*): this goal can be solved by several operators (*Operator-1*, *Operator-2*, and *Operator-3* in this case). A decomposition of a complex operator (*Operator-2*, or *Operator-3*) is a tree of requests to other subgoals. Each goal, and each operator have two private rule bases: one rule base on choice, and one on evaluation for a goal; one rule base on initialization, and one on adjustment for an operator.



- Figure 5 - Relations between the objects, and the rule bases in OCAPI

3.4.2 Implementation of the reasoning model

The control structures have five components : **a pilot, a planner, a parameter settler, a rule interpreter** (a general one called by the previous components), and **an interface to activate external programs.** The pilot has the role of a supervisor and of a controller: it controls the execution of a tree of requests, and decides to call other components for specific tasks. The algorithm presented in Figure 6 controls the execution of a tree of requests. Of course, to process the initial request, this algorithm works on a tree reduced to one request.

-1- global matching

while not all requests have been processed

-2- selection of the request to be processed

-3- classification of the operators (using choice rules)

while the request is not satisfied

-4- selection of the best operator

-5- execution of the operator (using initialization or adjustment rules)

-6- evaluation of the results (using evaluation rules)

- Figure 6 - The algorithm for the execution of a tree of requests

Step 1 (performed by **the planner**) is a global matching verifying that there exists at least one operator to solve the goals of the requests. Step 2 (performed by **the pilot**) selects one request to process (when the tree is reduced to one request, this step is immediate). Step 3 (performed by **the planner**) eliminates invalid operators, and performs a classification of the valid ones using the choice **rules.** Step 4 (performed by **the pilot**) selects one operator among the valid ones. Before execution of a chosen operator (step 5), **the parameter settler** determines the values of its parameters: using initialization **rules** for the first attempt, using adjustment **rules** for the next ones. If the operator is a primitive one, the execution is done by **the interface with programs**: in the case of a complex one, this algorithm is recursively called to control the execution of the tree of requests of its decomposition. Then, according to the request, the evaluation of the results (step 6) is achieved automatically using the evaluation **rules,** or by graphic presentation of the results to the user who evaluates them interactively. In case of unsatisfactory results, another execution is performed after adjustment of the input parameters of the operator, or after choice of another operator.

4 Examples of integration using OCAPI

In this section, we present two examples of expert systems built with OCAPI. The first example is the integration of a stereovisual process (including the extraction of the primitives); we show in detail how the various kinds of knowledge are expressed in the knowledge base. The second example is the description of a knowledge base developed for an application in astronomy; the integrated processing is a complex one.

4.1 Integration of a stereovisual processing

This example has been developed within the framework of the Eureka project Prometheus. It deals with detection of obstacles in road scenes and urban scenes, using stereovision data. The data are taken from two cameras which are fixed on the top of a moving car. A pyramidal stereovision algorithm based on contour chain points [Mey 90] is used to reconstruct the 3D environment in front of the vehicle.

Integration of the knowledge on the use of this stereovisual process is needed for robustness, and above all to manage the great variety of the scenes. More precisely a lot of contextual values in the scenes vary, like luminosity (depending on the weather conditions and on the moment in the day), complexity of the scene (depending on the number of objects in front of the car), and velocity (depending on the location of the scene: highway, countryside road or urban street). Since the scenes may be quite various, it is not possible to fix the values of the different parameters involved in the processing.

The initial request is thus the stereovisual processing of a pair of images taken by two cameras. Figure 7 shows an example of such images in the case of an urban scene.

The pyramidal process works at several resolutions (to be precise, four resolutions); for each resolution, first an extraction of the primitives, then a matching of these primitives are performed: for each image (the left and the right one), extraction of primitives consists of contour detection, thresholding by hysteresis, computation of precise orientation, and contour chaining. Matching results obtained at a given resolution are interpolated, and used to reduce the search at the immediately higher resolution.

- Figure 7 - Stereo images corresponding to an urban scene

4.1.1 Objects of the knowledge base

In this section, we give examples of the various objects of this knowledge base named PROMETHEE.

Context: The object **context** describes the possible values of the characteristics of the input data; this information is important because it is used in the rules (choice rules to decide pertinent operators in function of the context, initialization rules to adapt initial values of the parameters to the current context...). Presently, we use in the production rules several criteria as *nature of the objects* in the scene to decide which algorithm is well-adapted, eventual presence of *motion* in the scene (motion of the cameras and/or motion of the objects), level of *detail* we want to detect, amount of *noise* in the input image (this is related to the quality of the cameras). Some other characteristics (as *corners*) are related to specific programs which more particularly take into account these features. The *mode* characteristic specifies if the processing has to be performed in a completely automatic way (without possibility of asking a human user to validate the results) or in an interactive way.

The definition of the object **context** in this knowledge base is presented in Table 1.

CONTEXT ITEM	POSSIBLE VALUES	COMMENTS
nature-of-objects	*man-made, natural, mixed*	scene with natural, man-made, or both type of objects
noise	*low, important*	noisy image or not
motion	*present, absent*	presence or not of moving cameras or objects
details	*none, few, many, all*	need of more or less details
corners	*important, indifferent*	importance of these features
mode	*interactive, automatic*	interaction with the user or not

- Table 1 - Definition of the context in the PROMETHEE base

The initial request: We present in Figure 8 the initial request corresponding to the stereovisual processing of the images shown in Figure 7.

Request	**r_23**			
goal :	stereovision			
restrictions :	**input :**	imleft	**leftarrow**	*s8p10l.ext*
		imright	**leftarrow**	*s8p10r.ext*
		calleft	**leftarrow**	*calibl*
		calright	**leftarrow**	*calibr*
	output :	xyz	—	*s8p10.ext.xyz*
context :	nature-of-objects		*mixed*	
	motion		*present*	
	corners		*important*	
	noise		*low*	

- Figure 8 - The initial request *r-23*

This request has three main components: the name of its goal, the restrictions of the arguments of the goal (input and/or output), and its context. The name of the goal is *stereovision*. The restrictions of the input arguments set the input data (the left and right images *[s8p10l.ext* and *s8p10r.ext]*, and the corresponding calibration matrices *[calibl* and *calibr])*. The only restriction of the output arguments is to store the output 3D data in the file *s8p10.ext.xyz*. The particular context related to this request is as follows:

- objects can be both natural (because of the presence of pedestrians, cyclists, motorcyclists, trees, rocks) and man-made (because of the presence of vehicles, bridges, roads, buildings); objects in the latter category are usually well contrasted and can be easily described by polygonal approximation;
- there is *a priori* some motion in the scene because the car supporting the two cameras is moving, and mobile objects are usually present;
- corners are important features to detect especially for cars and trucks;
- the noise is low thanks to the quality of the acquisition process;

- the value of the user mode *interactive* or *automatic* is not specified.

Goals: This knowledge base contains several goals: *stereovision, contour-detection, chaining, consolidation, thresholding, sampling, orientation, stereo-primitives-extraction, primitives-extraction, initmatching, intermatching, finalmatching.* For instance, the first goal *stereovision* [Figure 9] has four input arguments (the pair of images, and the calibration matrices), and one output argument (the list of the 3D coordinates of matched primitives). Two choice rules are attached to this goal (R1 and R2) but there is not yet evaluation rules.

Goal	**stereovision**
input :	imleft
	imright
	calleft
	calright
output :	xyz
choice-rules :	{ R1, R2 }
evaluation-rules :	

- Figure 9 - The goal *stereovision* in the PROMETHEE base

Operators: Several operators have been defined in the knowledge base: they correspond to a particular goal as shown in Table 2.

goal	operators
stereovision	O-stereo-pyr1 , O-stereo-pyr2
contour-detection	O-deriche
chaining	O-chaining-gg1, O-chaining-gg2, O-anac-dzv
consolidation	O-reduc, O-id
thresholding	O-muls, O-hysteresis
sampling	O-sample
orientation	O-ori-grad
stereo-prim-extraction	O-ext-prims
primitives-extraction	O-ext-prim
initmatching	O-pyr-imatch
intermatching	O-pyr-match
finalmatching	O-pyr-fmatch

- Table 2 - The goals and their corresponding operators in the PROMETHEE base

For instance, the operator *O-stereo-pyr1* [Figure 10] which solves the goal *stereovision* has the same input arguments as its goal (the pair of images and the calibration matrices), and the same output argument (the list of the 3D matched primitives). Being a complex operator, it is decomposed into several substeps (requests to other goals): first extraction of primitives at a coarse resolution ($1/8 \times 1/8$) then a first matching, then the extraction

of primitives at the resolution $1/4 \times 1/4$, then matching, then extraction of primitives at resolution $1/2 \times 1/2$, then matching, then extraction of primitives at the highest resolution, then final matching.

Operator	**O-stereo-pyr1**
input :	imleft
	imright
	calleft
	calright
output :	xyz
decomposition :	request r8-ster-prim
	then request r8-initmatching
	then request r4-ster-prim
	then request r4-matching
	then request r2-ster-prim
	then request r2-matching
	then request r1-ster-prim
	then request r1-finalmatching
initialization-rules :	
adjustment-rules :	

- Figure 10 - The complex operator *O-stereo-pyr1* in the PROMETHEE base

Each step of the decomposition of the complex operator *O-stereo-pyr1* is a request; Figure 11 shows the details of the first one which is a request to the goal *stereo-prim-extraction* (the extraction of the primitives for stereovision) with a coarse resolution (resolution $1/8 \times 1/8$). Control of data flow of input and output data of the operator, and of the subrequests is expressed directly in the requests (as shown in Figure 11), and in the operator arguments (as shown in Figure 12).

Request	**r8-ster-prim**			
goal	stereo-prim-extraction			
restrictions :	**input :**	img	=	imleft of OPERATOR O-stereo-pyr1
		imd	=	imright of OPERATOR O-stereo-pyr1
		resolution	=	8

- Figure 11 - Description of the first subrequest of the operator *O-stereo-pyr1*

Argument	**xyz**	
	class	data
	direction	output
	type	list
	value	res-xyz of REQUEST r1-finalmatching

- Figure 12 - Description of the output argument *xyz* of the operator *O-stereo-pyr1*

4.1.2 Rules

In this section, examples of the various kinds of production rules present in the knowledge base are given.

Choice rules: When several operators are available to solve the same goal, we have to express how to select the pertinent operator according to the context. For example, as we have seen in the previous paragraph, two operators *O-stereo-pyr1* and *O-stereo-pyr2* can solve the goal *stereovision*. These two operators perform stereovision using a pyramidal technique, and are based on contour chain points. Thus they are well adapted to scenes with mixed type of objects (both natural and man-made); the first operator implements this algorithm, the second one begins with the separation of the two interlaced acquisition frames by doing a sampling of the lines. The second operator is thus well-adapted when there is motion in the scene, 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). Figure 13 shows *R1* one of the choice rules attached to the goal *stereovision*. Another rule (*R2*) exists to adapt the choice to the contexts of static scenes.

if	*type-of-objects mixed*
and	*motion present*
then	*:use-operator O-stereo-pyr2*
comment	"adapted to both mixed scenes only 1 of the 2 interlaced frames if motion"

- Figure 13 - The choice rule R1 attached to the goal *stereovision*

As we have seen in the previous paragraph, there are 3 operators which can solve the goal *chaining*: two of them are programs (the operators *O-chaining-gg1*, and *O-chaining-gg2*), the last one *O-anac-dzr* is a complex operator which is a decomposition into two subrequests. Figure 14 shows that we use in priority the operator *O-chaining-gg1* if the corners are important features in the scene.

if	*corners important*
then	*:use-operator O-chaining-gg1*
comment	"preserve the corners and the isolated points"

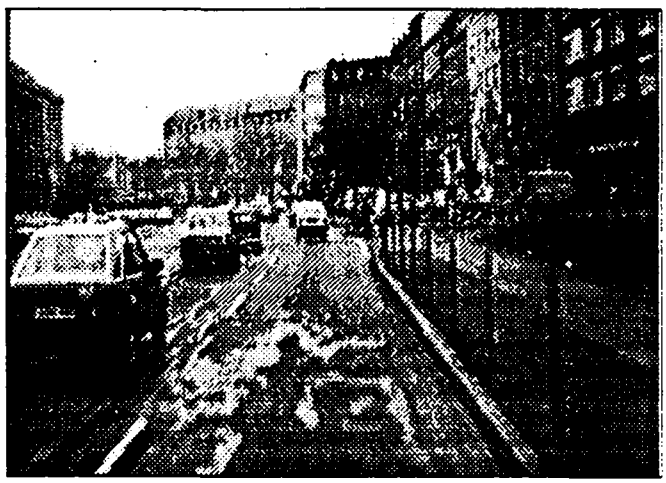
- Figure 14 - The choice rule R4 attached to the goal *chaining*

Evaluation rules: Most of the time it is very difficult to express evaluation criteria to assess the results of low level goals (like thresholding) since, for example, results are matrices of pixels; nevertheless when the system is used in an interactive mode, it is very convenient to display to the user the output images, and a list of possible predefined judgements. For example, a thresholding by hysteresis operator is used for the extraction

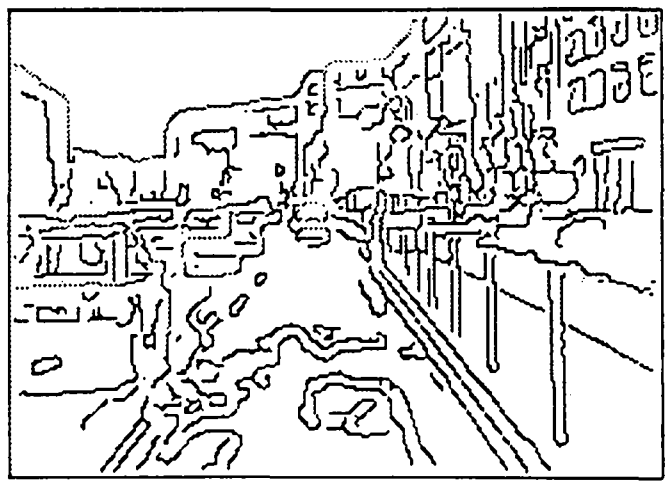
of primitives: the system needs to know if the results of the thresholding are correct, or not in order to be able to continue the execution. Figure 15 shows the rule which, depending on the mode of use, performs a query to the user. Figure 16 shows the effect of the activation of this rule.

```
if          mode interactive
then        :assess-by-user detection (correct, noisy, insufficient)
comment     "if interactivity ask the user for the quality of the detection"
```

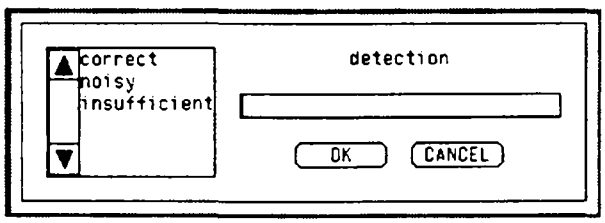
- Figure 15 - One evaluation rule for the goal *thresholding*



input image



output image



- Figure 16 - Evaluation of the results of a thresholding by interaction with the user

When a goal provides high level data as results, like some measure, it is possible to find a criterion of evaluation. For example, the intermediate level goal *matching* provides as output matched pairs of primitives which can be counted and compared to a required value. The rule shown in Figure 17 tests the ratio between the number of matched primitives to the number of input primitives; if this ratio is below a certain value (1/2) the rule qualifies the number of matched pairs of primitives as insufficient, and a failure is decided.

```
if      nb_pairs < nb_primitives/2
then    :assess nb_pairs insufficient
        :failure
```

- Figure 17 - One automatic evaluation rule for the goal *matching*

Initialization rules: Initialization of the parameters of the operators depends highly on the context. For instance, the operator *O-hysteresis* which performs a thresholding by hysteresis needs the initialization of two input parameters: a high threshold (*thrmax*) and a low threshold (*thrmin*). These input arguments are defined as parameters which are numerical real values within a range of -10^5 to 10^5, and a default value of 128 for *thrmax*, and 10 for *thrmin*. But these values depend on the range of the image (minimum, average and maximum values), and on the quantity of details required in the request. So 8 initialization rules are needed to define the two thresholds in function of the context of use of the operator. For example Figure 18 shows initialization of *thrmax*, and *thrmin* when only few details are requested.

```
if          details few
then        :initialize thrmax average + (maximum - average)/2
            :initialize thrmin thrmax/2
comment     "usual values for high and low thresholds"
```

- Figure 18 - One initialization rule of the operator *O-hysteresis*

Adjustment rules: When an operator fails because the results are not satisfactory, we need to express how to modify its parameters in function of the type of the failure. For example, we have seen in the two previous paragraphs that: first, the result of the thresholding by hysteresis is assessed interactively by the user, and the judgement is set to one of the possible values among *correct, noisy*, or *insufficient;* second, the operator *O-hysteresis*, which performs a thresholding by hysteresis, has two input parameters: *thrmax* and *thrmin*. If the user estimates that the detection is not correct, the system can decide to use an adjustment method by dichotomy for the parameter *thrmax*, for instance, and to specify that the range will be the minimum and maximum values of the image. If, for instance, the user has assessed that the detection was *noisy*, the parameter *thrmax* has to be increased. Figure 19 shows adjustment rules for this case.

```
if      ...
then    :adjust-by dichotomy thrmax
        :dich-min minimum
        :dich-max maximum

if      :assessed? detection noisy
then    :increase thrmax
```

- Figure 19 - Two adjustment rules of the operator *O-hysteresis*

In the same way, we have seen that the evaluation of the goal *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 operator *O-pyr-matching* which are: a threshold on the orientation of the gradient *(thr-o)*, and a threshold on the magnitude of the gradient *(thr-m)*: one adjustment rule for this operator is shown in Figure 20.

```
if      ...
then    :adjust-by percentage thr-o
        :%-step thr-o .3

if      :assessed? nb_pairs insufficient
then    :increase thr-o
        :increase thr-m
```

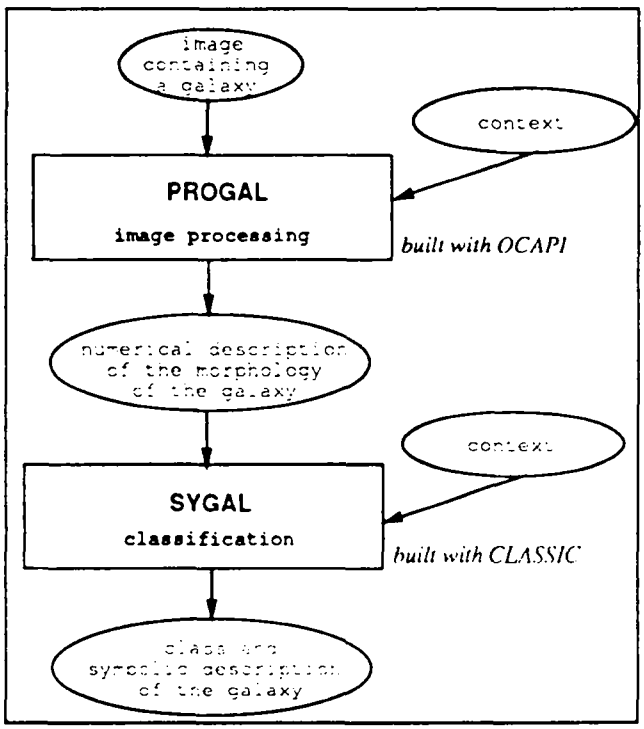
- Figure 20 - Two adjustment rules of the operator *O-pyr-matching*

4.1.3 Conclusion

The current knowledge base is composed of 15 goals, at which are attached 19 operators. Among these operators, 14 are primitive ones (programs) and 5 are complex ones (with decomposition into steps). 50 production rules are attached to those frames: there are 6 choice rules, and 10 evaluation rules attached to the goals; 14 adjustment rules, and 20 initialization rules are attached to the operators. Thanks to 34 inferences rules for initialization and adjustment of parameters, this knowledge base allows to process fully automatically really different types of data. This knowledge base could be extended by adding new alternative operators both for extraction of primitives and matching, with their corresponding rules.

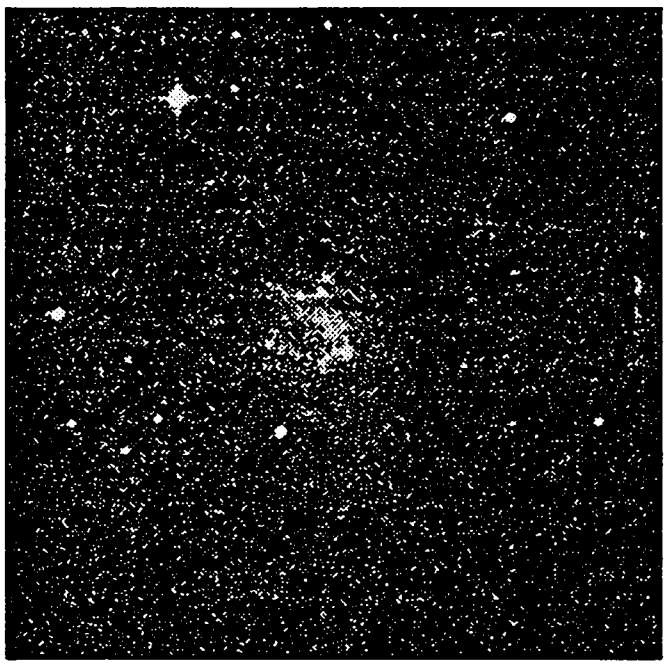
4.2 PROGAL: an integration in astronomy

The problem of integrating image processing modules arises also for the development of specific applications. It is the case for applications which have to be flexible, i.e. the processing must be adaptable according to changes in the environment or to the type of data. The example presented here deals with the automatization of a processing of images containing a galaxy. The role of this processing is to detect, isolate and describe the galaxy before its classification by the expert system SYGAL [Tho 89] built with CLASSIC [Bro 87], a classification tool generator. A synopsis of the processing is shown in Figure 21.



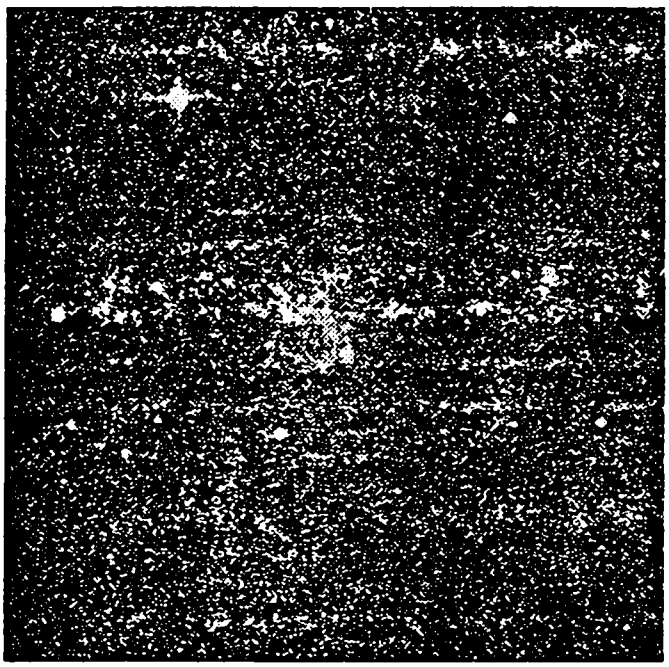
- Figure 21 - Synopsis of the global processing of the images containing a galaxy

So, for each image, the goal of the image processing is the same: to describe the morphology of the galaxy in terms of numerical parameters. For example, Figure 22 shows the input image *mt671n17.fl* containing in the center the galaxy *ngc4523*, and the numerical description of the galaxy computed by the processing. But as there is a great variability in the images (some examples are shown in Figure 23), both the sequences of procedures, and the values of their parameters need to be adapted. Using OCAPI, we have built the PROGAL expert system to pilot the processing.

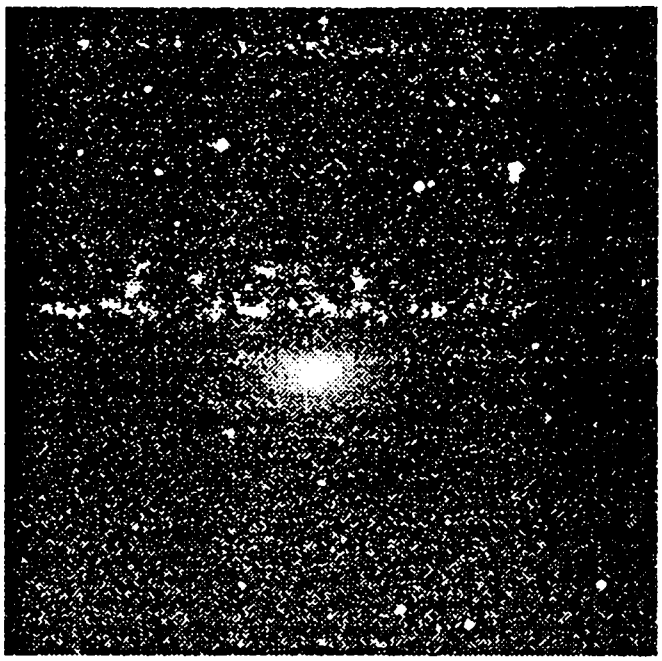


```
galaxy :  ngc4523 :
          area              : 4536.5 ;
          ellipticity       : 0.76 ;
          linear_err        : 0.469 ;
          profile           : 1.08 ;
          orientation       : 53.74 ;
          contour c1 : {  centre_err      : 2 ;
                          ellipse_err     : 0.01 ;
                          compactness     : 1. ;
                          angle           : -3.4 ;
                          eccentricity    : 0.62 };
          contour c2 : {  centre_err      : 2 ;
                          ellipse_err     : 0.16 ;
                          compactness     : 1.4 ;
                          angle           : -10.1 ;
                          eccentricity    : 0.26 };
          contour c3 : {  centre_err      : 2 ;
                          ellipse_err     : 0.11 ;
                          compactness     : 2.4 ;
                          angle           : -25.1 ;
                          eccentricity    : -0.01 };
          contour c4 : {  centre_err      : 34 ;
                          ellipse_err     : 0.47 ;
                          compactness     : 13.8 ;
                          angle           : 47.1 ;
                          eccentricity    : 0.27 };
          contour c5 : {  centre_err      : 19 ;
                          ellipse_err     : 0.64 ;
                          compactness     : 15.1 ;
                          angle           : -88.1 ;
                          eccentricity    : 0.4 }.
```

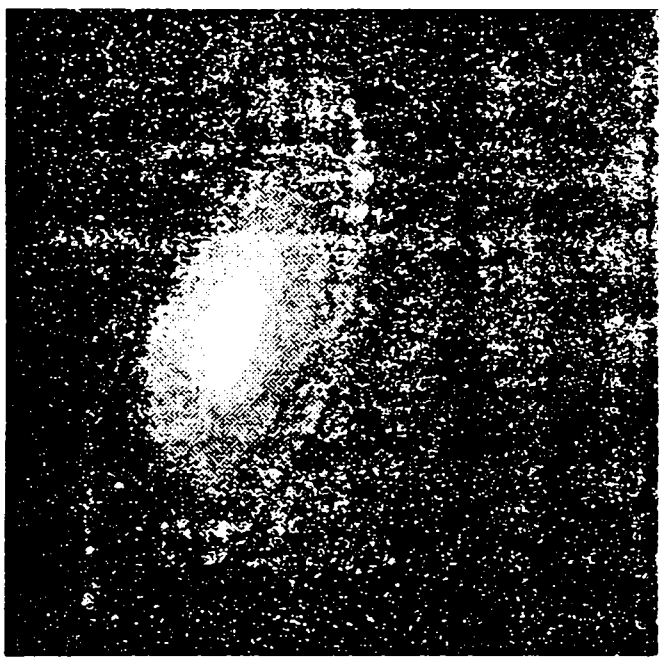
- Figure 22 - The input image *mt671n17.fl*, and the results of its processing



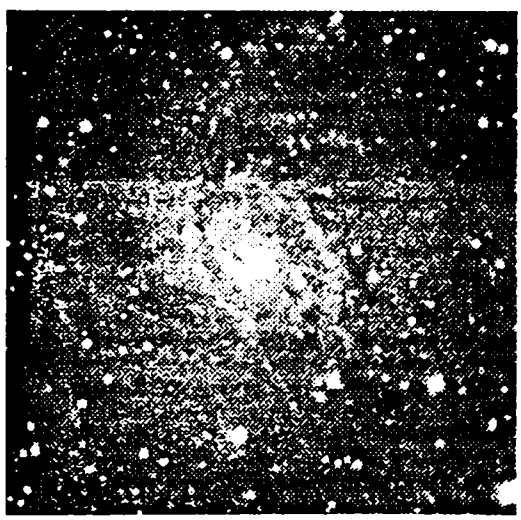
ngc4523



ngc4473



ngc7531



ngc6946

- Figure 23 - Some images of galaxies

In this application, the set of procedures to pilot is composed of specific procedures (C and Fortran77), standard image processing procedures of a library (INRIMAGE), and operating system commands (Unix). Some procedures such as histogram computing, filterings (median, morphological), extraction of images, and convolutions are standard ones. Specialized algorithms are, for example, computation of some thresholds, isolation of the galaxy, or computation of morphological parameters.

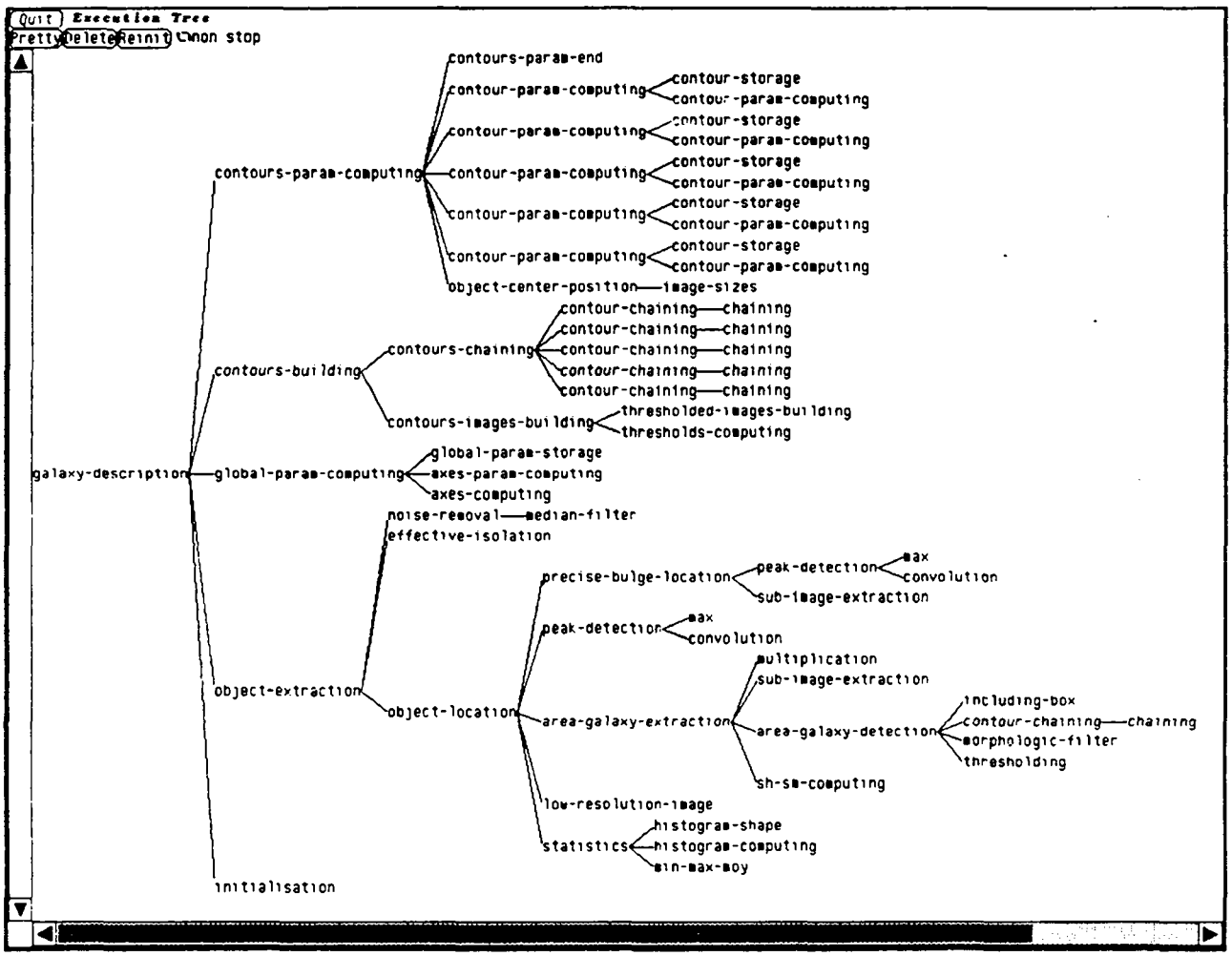
In fact, there are low or high resolution images, noisy or good images, CCD or photographic plates, centered or uncentered galaxy on the image, eventual presence of bright foreground stars. So, in this application, the context of the knowledge base takes into account these criteria, as it is shown in Table 3.

CONTEXT ITEM	POSSIBLE VALUES	COMMENTS
galaxy-position	*centered uncentered*	position of the object in the image
min-size	[1, 1000000]	minimal size of the object expressed in pixels (fonction of the observation instrumentation)
noise	*low important*	noisy image or not
image-type	*intensity density*	image type
stars	*in-front-of-the-galaxy absent everywhere outside-the-galaxy*	eventual presence of other objects

- Table 3 - Definition of the context in the PROGAL base

The structure of a complete processing executed by PROGAL is presented in Figure 24. The initial request refers to the most general goal of the knowledge base, which is *galaxy-description*, on the image *mt671n17.fl* which contains the galaxy *ngc4523*. The tree expresses the decomposition of the final processing into substeps up to primitives operators (programs). The level of abstraction is represented on the horizontal axis: from the most abstract on the left, to the least one (the concrete programs on the right). The temporal ordering is represented on the vertical axis: the lowest leaf corresponds to the first step, the uppermost one to the last step. In this figure, nodes are goals to be solved.

During this execution 49 programs are run; among them 4 programs are executed 3 times each, during the evaluation-adjustment phases. Figure 25 shows the sequences of the commands sent to the image processing system. For the execution of this request, choice rules and evaluation rules have used contextual information about the position of the galaxy in the image (centered or not), the presence of noise, the presence of stars, and the type of the images (density or intensity). As this information was not specified to the system in the initial request, the user has been asked for during the execution.



Left to right: from the initial request to concrete programs.
Bottom to top: from the first substep to the last substep.

- Figure 24 - A complete execution tree of the PROGAL base

```
cal mt671n17.fl >inter.cal
his mt671n17.fl histogramme
hseuil histogramme >inter.cs
sample -ex 4 -ey 4 mt671n17.fl mt671n17.fl.sam
muls -vs .2728446 mt671n17.fl.sam mt671n17.fl.sam.muls
morphlis mt671n17.fl.sam.muls 1
anac -fc mt671n17.fl.sam.muls.l mt671n17.fl.sam.ch
boite mt671n17.fl.sam.ch mt671n17.fl.sam.boi >boite.r
muls -vs .2592023 mt671n17.fl.sam mt671n17.fl.sam.muls
morphlis mt671n17.fl.sam.muls 1
anac -fc mt671n17.fl.sam.muls.l mt671n17.fl.sam.ch
boite mt671n17.fl.sam.ch mt671n17.fl.sam.boi >boite.r
muls -vs .2509509 mt671n17.fl.sam mt671n17.fl.sam.muls
morphlis mt671n17.fl.sam.muls 1
anac -fc mt671n17.fl.sam.muls.l mt671n17.fl.sam.ch
boite mt671n17.fl.sam.ch mt671n17.fl.sam.boi >boite.r
ext -ix 62 -iy 61 -x 15 -y 15 mt671n17.fl.sam mt671n17.fl.sam.ext
mu mt671n17.fl.sam.ext mt671n17.fl.sam.boi mt671n17.fl.sam.zc
convo mt671n17.fl.sam.zc noyau mt671n17.fl.sam.zc.conv
dmax mt671n17.fl.sam.zc.conv >mt671n17.fl.sam.zc.conv.max
ext -ix 248 -iy 248 -x 21 -y 21 mt671n17.fl mt671n17.fl.ext
convo mt671n17.fl.ext noyau mt671n17.fl.ext.conv
dmax mt671n17.fl.ext.conv >mt671n17.fl.ext.conv.max
isole1 mt671n17.fl mt671n17.fl.c mt671n17.fl.g -x 267 -y 268 -s1 .09999998 -s2 .01 -sh .3357878
        -sm .2400045 >> inter.is
medmt 3 mt671n17.fl.g mt671n17.fl.g.med
axes mt671n17.fl.g.med mt671n17.fl.g.med.c2 >inter.axes
calaxes -th 53.74 mt671n17.fl.g.med mt671n17.fl.c mt671n17.fl.g.med.c2 mt671n17.fl.g.med.c7 >inter.cpa
seuils mt671n17.fl.g.med -s1 .202636 -s2 .156246 -s3 .04896 -s4 .022766 -s5 .013338 contour/c1mt671n17.fl
        contour/c2mt671n17.fl contour/c3mt671n17.fl contour/c4mt671n17.fl contour/c5mt671n17.fl
anac -fc contour/c1mt671n17.fl contour/c1mt671n17.fl.r
anac -fc contour/c2mt671n17.fl contour/c2mt671n17.fl.r
anac -fc contour/c3mt671n17.fl contour/c3mt671n17.fl.r
anac -fc contour/c4mt671n17.fl contour/c4mt671n17.fl.r
anac -fc contour/c5mt671n17.fl contour/c5mt671n17.fl.r
par mt671n17.fl.g.med >inter.par
calcont2 contour/tek.mt671n17.fl contour/c1mt671n17.fl.r 32 >inter.cpc
calcont2 contour/tek.mt671n17.fl contour/c2mt671n17.fl.r 32 >inter.cpc
calcont2 contour/tek.mt671n17.fl contour/c3mt671n17.fl.r 32 >inter.cpc
calcont2 contour/tek.mt671n17.fl contour/c4mt671n17.fl.r 32 >inter.cpc
calcont2 contour/tek.mt671n17.fl contour/c5mt671n17.fl.r 32 >inter.cpc
clearf contour/tek.mt671n17.fl contour/tekn.mt671n17.fl
```

- Figure 25 - Image processing performed on the image *mt671n17.fl*

Only one operator is attached to the goal *galaxy-description*. It corresponds to the global processing to perform on each image; its decomposition into steps is as follows: initialization, isolation of the galaxy (goal *object-extraction*), computation of global parameters of the galaxy (goal *global-param-computing*), building of images of contours (goal *contours-building*), and then, for each contour, computation of parameters (goal *contours-param-computing*). The four last steps are themselves complex.

One step of isolation of the galaxy is its localization (goal *object-location*), which is decomposed into 5 steps: computation of density statistics, building of a low resolution image, extraction of the area of the galaxy (which can be done in different ways), approximative localization of the bulge (goal *peak-detection*), and then precise localization of the bulge.

Choice rules determine which operator has to be used to extract the area of the galaxy (depending if the galaxy is centered or not in the image), and if noise reduction has to be done. These rules use information contained in the contextual base of facts.

The thresholds are initialized using default values, and by means of rules depending on the context, and on values related to other parameters.

The evaluation of the area galaxy extraction is done automatically. In fact, the evaluation is based on the size of the detected object as expressed in the request *r-agd* (see Figure 26) which is one of the subrequests of the operator *O-uncentered-gal* for the area galaxy extraction. The restriction on the output argument *size* expressed in this request is based on the fact that the galaxy is supposed to be greater than a minimal value which depends on the context. So, the evaluation is done at a high level of abstraction; this can involve another execution of a low level operator (a thresholding operator) with another value of parameter (*thr*, a threshold). In this case, an evaluation rule is : "if size-of-the-detected-object too-small, then detection insufficient"; an adjustment rule is: "if detection insufficient, then decrease *thr*."

Request:	**r-agd**			
goal	area-galaxy-detection			
restrictions :	**input :**	ie	=	ie of OPERATOR O-uncentered-gal
		sm	=	sm-mod of REQUEST r-css
	output :	size	>	:context min-size

- Figure 26 - Request *r-agd* to perform the area galaxy detection

The current knowledge base is composed of 44 goals, at which are attached 47 operators. Among these operators, 28 are primitive ones (programs) and 19 are complex ones (decomposition into steps). 21 rules are attached to those frames. This example shows that it is possible to develop, with OCAPI, knowledge bases for specific applications, with a complex structure and numerous steps.

4.3 Building knowledge bases in integration

From the two examples previously described, we can deduce several points concerning the building of such knowledge bases in integration. For all kinds of expert systems (diagnosis, planning...), the building of the knowledge base must begin with the description of the static knowledge: first, this knowledge is the most stable; second, this knowledge is the support for dynamic knowledge. In the case of integration, the process has to be the same: static knowledge being the context, the goals, the operators, and the requests, and dynamic knowledge being the choice, evaluation, initialization, and adjustment rules.

Static knowledge: Developing PROMETHEE and PROGAL presented in the previous sections, has shown the necessity of beginning with the description of all available programs, which, in fact, are the basic building blocks of the knowledge. So frames describing the primitive operators are the first to be expressed. Next step is developing of higher levels of abstraction operators; depending on applications, particular sequences of processing can be expressed.

Dynamic knowledge: Developing of the dynamic knowledge is a more complex process; it may imply developing new algorithms for, for example, initialization of parameters or evaluation of results. It may also imply the extension of the knowledge base or modifications of some programs because, at this step, the expert usually will be conscious that a great deal of knowledge is implicit inside the existing programs; so, he or she will try to make them explicit in the knowledge base.

5 Discussion

We now briefly describe the general mechanisms of the tools presented in section 2, and summarize those of OCAPI. Then we show the concepts which are emerging from all this work.

5.1 Mechanisms

We discuss now the knowledge representation mechanisms, and the reasoning mechanisms provided by these systems. We will also distinguish between knowledge which can be incorporated to the system from the formalisms used to express it.

5.1.1 Expert assistant

The Expert assistant system [Joh 87] allows to represent the knowledge of different kinds of data, instrument modes, and data analysis operations expressed in term of tasks. Tasks

can be primitive or compound. Primitive tasks are those which can be implemented with a single command. Compound tasks represent higher-level operations; they cannot be directly converted into analysis commands, but must be expanded into a network of subtasks. Control is done by rules, which reasons about data properties, merges redundant tasks, and expands compound tasks. First, a symbolic plan is generated. Then, this plan is translated into commands for one of the two available image analysis systems.

5.1.2 Toriu's system

Given a predefined goal, and a symbolic description of the attributes of the image, this system [Tor 87] selects a global processing, and image processing modules; it also runs them. A distinction is made between abstract goal, decomposition into subgoals, and programs. Programs are explicitly described, using frames, by their preconditions, and effects on the characteristics of the images. Planning is performed by production rules (260) either for the decomposition of the goal into subgoals, or for the selection of the programs. The system can ask the user to qualify the input data (initial or intermediate images) in order to process them; but this is not what we call an evaluation because it is not on the quality of the results (output data). This system is highly dependent on the image processing system (FIVIS).

5.1.3 EXPLAIN

In EXPLAIN [Tan 88], knowledge is described by a set of production rules. A first set of production rules selects sequences of process to reach the goal. A decomposition into sequences, which is considered to be subgoaling, is done using the depth first search strategy; when a subgoal corresponding directly to an actual algorithm, is reached, the image processing subsystem is called. Executable commands is also generated by activation of rules. Evaluation is interactive. Adjustment of the parameters of an operator, and change of operator are done by other production rules.

5.1.4 DIA-ES

Concerning DIA-ES [Tam 84], we only describe its semantics processor, the syntactical processor corresponds to another objective (generation of code). Knowledge in DIA-ES is expressed using several levels of abstraction. The planning phase includes matching of goal, and choice rules; it enables some dynamic modifications in a decomposition. Each command is described in the system by a frame. An execution tree, built by production rules, describes the processing being elaborated. These rules express the decomposition of the goal into subgoals, at different levels of abstraction or modifications in the structure

of the execution tree (suppression, insertion of new operations). Manipulating the execution tree allows automatic adjustment of the processing. This tool is highly interactive. Evaluation of results, and adjustment of parameters are performed by the user.

5.1.5 Bailey's system

In the prototype defined by Bailey [Bai 88b], notions of goal, operator, operator selection, and decomposition are present. Selection is performed through dialog with the user; decomposition does not include several possible levels of abstraction. It is interesting to observe that initialization of parameters can be done through recursive calls to the system; but there is no adjustment of the parameters, because there is no loop mechanism allowing repeated executions of a same operator for the same task. Evaluation which is associated to the operator (and not to the goal) is done through dialog with the user. In case of unsatisfactory results, another sequence of operators will be used.

5.1.6 OCAPI

In OCAPI, the formalisms used to express the knowledge are both frames and rules. The main kinds of knowledge handled by the system are: context (description of the data, condition of acquisition, application domain...), image processing functionality (goals), and primitive or complex operator (actions which can be performed). The reasoning (planning, and control of execution), is controlled by decompositions expressed in complex operators, and different kinds of rules (choice, evaluation, initialization, and adjustment).

5.2 Emerging concepts

From the presentations above, we can notice some emerging concepts. We will discuss them according to the description of the static knowledge, to the planning facilities, and to the characteristics of the control of execution.

- Description of the static knowledge:
  - the notion of *multiple levels of abstraction* of the goals, and of the operators of image processing is taken into account by all the systems. However, there are some differences. Bailey's system and Expert assistant only distinguish between decomposable operators, and programs (in fact, Bailey's system has a notion of goal, but only at one level of abstraction). DIA-ES, EXPLAIN, Toriu's system, and OCAPI introduce the notion of abstract goals, with operators which are decomposable. However, in OCAPI, decomposition of an operator is expressed using requests instead of subgoals: in fact, a request allows to manage the data flow, and to express constraints on the results.

- *description of operators* has to be explicit. An operator (complex or primitive one) is an entity of knowledge which can be described in a static way: by the list of its arguments (data and/or parameters), its preconditions and/or effects, the way to run it (decomposition or syntax). Bailey's system, DIA-ES, and OCAPI allow to express this kind of information, while, in Toriu's system, operators are described only by their preconditions and effects. In Expert assistant, only primitive operators have an explicit representation.
- an expert in image processing knows which characteristics of an image are crucial to determine how to process it. So, description of the images with symbolic attributes is generally used (EXPLAIN, and Toriu's system). This can be extended to the notion of context (Expert assistant, and OCAPI) allowing to express knowledge about the world: nature of the acquisition instruments, or type of the observed scene, for instance.

• Planning facilities:

- a common objective of all these systems is to automate the *selection of the operators* (complex or primitive). In fact, this is done through inferences. It can also be done using automatic matching on preconditions or effects (Bailey's system). For systems with facilities for control of execution, in case of a failure of the selected operator, a backtrack mechanism is provided (EXPLAIN, Bailey's system and OCAPI) to select another operator to solve the current goal.
- in image processing, very often, the expert has in mind, at various levels of abstraction, a skeletal plan containing the basic steps to solve the problem. This is the concept of decomposition by *hierarchical skeleton*. It can be implemented using production rules (Expert assistant, EXPLAIN, and Toriu's system), or using frames (Bailey's system, DIA-ES, and OCAPI). It is important to observe that it provides also, especially in the second case, an effective saving of computing time.
- the concept of decomposition by hierarchical skeleton is essential; but, in order to have a fully flexible behavior, a mechanism allowing case-specific and *dynamic modifications on the skeleton* is needed. In DIA-ES, insertion or deletion of operations can be done using inferences.

• Characteristics of the control of execution: *(Reminder: Expert assistant and Toriu's system have no facilities for control of execution.)*

- the *initialization of the parameters* are done through interaction with the user (DIA-ES, and OCAPI), using inferences (Bailey's system, DIA-ES, EXPLAIN, and OCAPI), or with recursive calls (Bailey's system, and OCAPI).

- *evaluation of the results* is a more difficult task. So in some systems, this is only done by interaction with the user (Bailey's system, DIA-ES, EXPLAIN, and Toriu's system). In addition, in OCAPI, when available, knowledge of evaluation of results is expressed using rules.
- repeated executions of a same operator can be needed to obtain satisfactory intermediate or final results: this is done by a *loop mechanism* (DIA-ES, EXPLAIN, and OCAPI). Of course, the notion of *adjustment of parameters* is taken into account only by these systems. It is done through interaction with the user (DIA-ES, and OCAPI), or automatically (EXPLAIN, and OCAPI).

There is not yet a system including all these concepts. More particularly, OCAPI, which includes most of them does not yet have facilities for dynamic modification of the skeleton, as DIA-ES has. For instance, to perform such modification could be useful for facultative operations as filtering for noise removal. On the other hand, OCAPI offers the most developed trial-and-error mechanisms (evaluation of results, and adjustment of parameters).

One can notice that none of these systems has implemented the notion of hierarchical classification of goals or methods; this feature could, nevertheless, be very important for expressing knowledge in a natural way, and help structuring big knowledge bases. For instance, using the terminology defined in OCAPI, first the goal *segmentation* could be described, then subclasses could be added to this goal: subclass *contour-based segmentation*, subclass *region-based segmentation*, and subclass *mixed-contour-region segmentation*. Another example can be given in stereovision: first the goal *stereovision* could be defined, then subclasses *three-cameras stereovision* and *two-cameras stereovision* could be added. Of course several operators (complex or primitive ones) may exist for each of these subclasses.

6 Conclusion

We have introduced the notion of semantic integration of programs. Then, a model of reasoning and knowledge involved in image processing has been proposed. An implementation of this model, OCAPI, which has the architecture of an expert system generator, has been described.

Two examples of utilization of OCAPI have been shown: a knowledge base on a stereovisual process, and another one for morphological description of galaxies. Currently, several other knowledge bases are under development on various domains, such as on stellar astrophysics (planning and control of execution of numerical algorithms), and on remote sensing (an industrial application).

Various other tools allowing semantic integration of programs have been discussed. Although, knowledge representation and control structures are very rich within the OCAPI system, we plan to extend the model in two ways: first, extension of the planning facilities by allowing dynamic modification of the skeleton; second, addition, to the knowledge model, of a hierarchical classification of goals.

References

[Bai 88a] D.G.Bailey and R.M.Hodgson, VIPS - a digital image processing algorithm development environment, *Image and Vision Computing* **6.3**, 176-184, Butterworth and Co. (Publishers) Ltd, 1988.

[Bai 88b] D.G.Bailey, Research on computer-assisted generation of image processing algorithms, in *Processdings, IAPR Workshop on Computer Vision - Special Hardware and Industrial Applications*, Tokyo, 1988, pp.294-297.

[Bar 82] A.Barr, P.R.Cohen and E.A.Feigenbaum, *Handbook of Artificial Intelligence*, Pitman, 1982.

[Bro 81] R.A.Brooks, Symbolic Reasoning Among 3-D Models and 2-D Images, *Artificial Intelligence Journal* **17**, 1981, 285-348.

[Cle 89] V.Clément and M.Thonnat, Handling knowledge on image processing libraries to build automatic systems, in *Proceedings, Int. Workshop on Industrial Applications of Machine Intelligence and Vision*, Tokyo, 1989, pp.187-192.

[Han 87] A.Hanson and E.Riseman, The VISIONS image-understanding system, in *Advances in Computer Vision*, (C.M.Brown, Ed.), pp.1-114, Erlbaum Assoc, 1987.

[Hen 90] J.Hendler, A.Tate and M.Drummond, AI Planning: Systems and Techniques, *AI Magazine* **Summer 1990**, pp.61-77.

[Ike 88] K.Ikeuchi and T.Kanade, Automatic Generation of Object Recognition Programs, in *Proceedings of the IEEE* **78.8**, 1988, pp.1016-1035.

[Ilo 87] ILOG, CLASSIC Manuel de l'utilisateur, Paris, 1987.

[Int 85] Intellicorp, *KEE System Manual*, Menlo Park, CA, 1985.

[Joh 87] M.D.Johnston, An expert system approach to astronomical data analysis, in *Proceedings, Goddard Conf. on Space Applications of Artificial Intelligence and Robotics*, 1987, pp.1-17.

[Mat 89] T.Matsuyama, Expert systems for image processing: Knowledged-based composition of image analysis processes, *Comput. Vision Graphics Image Process.* **48**, 1989, 22-49.

[Mey 90] A. Meygret, M. Thonnat and M. Berthod, A pyramidal stereovision algorithm based on contour chain points, in *Lecture Notes in Computer Science* **427**, (O.D.Faugeras, Ed.), pp.83-88, Springer-Verlag, 1990.

[Naz 84] A.M.Nazif and M.D.Levine, Low Level Image Segmentation: An Expert System, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6.5**, 1984, 555-577.

[Sak 85] K.Sakaue and H.Tamura, Automatic generation of image processing programs by knowledge-based verification, in *Proceedings, IEEE on Computer Vision and Pattern Recognition*, San Francisco, 1985, pp.189-192.

[Sat 88] 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*, Tokyo, 1988, pp.413-416.

[Tam 83] H.Tamura *et al.*, Design and implementation of SPIDER - a transportable image processing software package, *Comput. Vision Graphics Image Process.* **23**, 1983, 273-294.

[Tam 84] H.Tamura and K.Sakaue, DIA (Digital Image Analysis) - Expert System : an approach to future vision system design, in *Int. Symp. on Image Processing and its Applications*, Tokyo, 1984.

[Tan 88] T.Tanaka and N.Sueda, Knowledge acquisition in image processing expert system EXPLAIN, in *Proceedings, Int. Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, 1988, pp.267-272.

[Tho 88] M.Thonnat and M.H.Gandelin, An expert system for the automatic classification and description of zooplanktons from monocular images, in *Proceedings, 9th Int. Conf. on Pattern Recognition, Roma*, 1988, pp.111-118.

[Tho 89] M.Thonnat and A.Bijaoui, Knowledge-Based Classification of Galaxies, in *Knowledge-Based Systems in Astronomy* (F.Murtagh and A.Heck Eds), pp.121-159, Springer-Verlag, 1989.

[Tor 87] T.Toriu, H.Iwase and M.Yoshida, An Expert System for Image Processing. *FUJITSU Sci.Tech.J.* **23.2**, 1987, 111-118.

[Vog 86] R.C.Vogt, Formalized Approaches to Image Algorithm Development Using Mathematical Morphology, in *Proceedings, VISION'86*. Detroit, 1986.

Contents

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249 - 6399