# Behaviour-Oriented Commands: From Distributed Knowledge Representation to Real-Time Implementation

J.Cuervo, E. González*, A. Suárez*, C. Moreno, F. Artigue

Université d'Evry Val d'Essonne - CERMA, CEMIF, LaMI
40, Rue du Pelvoux CE 1455 Courcouronnes 91020 Evry Cedex FRANCE

## Abstract

*This paper presents a general methodology to model and implement real time control of complex systems with high reactivity. It is based on an original concept called "behaviour oriented commands" (BOCs). This methodology has been applied successfully in our mobile robot.*

*BOCs incorporate mechanisms to model the set of rules (knowledge) which describes the restrictions and actions to achieve a goal. Basic rules are well encapsulated by entities called "behaviours", while global co-operating rules are explicited by the association links managed by the BOC's control unit. The model is easily translated into a real time implementation. This fusion between knowledge and real time is the main contribution of our work to the RT. area.*

## 1. Introduction

We are interested in the coexistence between real time distributed systems and dynamic knowledge management. This paper presents a mobile robot application, implemented on a multiprocessor real time architecture. The robot evolves in a totally unknown indoor environment while acquiring information. This implies for the mobile robot a permanent interaction. Autonomy is demonstrated by imposing to the robot as mission to explore and to navigate the entire accessible area [SUA95]. The robot's limitations on hardware and computational power are successfully overcome by means of a software parallelism and a distributed architecture involving several CISC type processors.

In complex systems, like mobile robots, a large amount of independent activities are needed in parallel to assure reactivity and correct timing of the action execution [ELF86]. The *formalisation* of this activities results in a set of concurrent tasks. Event's handling, communication and task management mechanisms are required in order to guarantee adequate time and space responses.

Systems of this type, where information, control and decision levels must coexist, need a knowledge management mechanism that assures its coherence within all distributed modules. In order to solve these constraints, we support our work on Distributed Artificial Intelligence (DAI) concepts [DEC87]. We developed a behaviour based robot control architecture in order to achieve a natural design and management of the actions needed to accomplish the robot's mission.

Our "behaviour oriented commands" (BOCs) define a software architecture. They require a set of operation rules that must be respected by the system (mobile robot) in order to accomplish its goal. Validation has been done by verifying the coherence of the system behaviour in accordance with the defined set of rules. A real time implementation can be directly obtained from the BOCs model.

## 2.Distributed Artificial Intelligence Approach

Our work is based on a systemic approach [BRU94], where knowledge is distributed among the processing entities that compose our application. Each must have their own information, decision and control systems in order to be considered as autonomous intelligent subsystem. The intelligence of our system is related to the establishment of association links between the distinct entities, in order to realise the different actions at the right moment.

The ability to act properly in unknown and dynamic environments is critical for all living beings. Primitive

1068-3070/96 $5.00 © 1996 IEEE
*Proceedings of EURWRTS '96*

species survive by using only pure reactive mechanisms (direct answer to stimuli) while advanced species need to anticipate future events and make plans of actions to carry out their goals [GEO87]. In order to design intelligent autonomous robots, it is necessary to include mechanisms which guarantee both the coherent execution of plans and event reactivity. In order to incorporate the reactive aspects we exploit some concepts of the "behaviour" architecture proposed by Brooks [BRO86]. Our architecture uses a top-down approach of communicant system processes (C.S.P.) where the design methodology is supported by two classical notions: abstraction and decomposition.

Abstraction allows us to define a general entity without knowing its implementation while decomposition permits to model complex systems as a set of less complex subsystems. We applied these concepts to model "behaviours", treatments and data. For example a complex behaviour is decomposed iteratively until "elementary behaviours" are reached. An "elementary behaviour" is an entity which interacts directly with the physical environment.

Decomposition criteria applied to find the net of (C.S.P.) are the modularity and the parallelism of behaviours. This "behavioural" decomposition of the system allows the simultaneous representation of both the temporal evolution of behaviours, and the parallel relation of the treatments.

This behaviour inspired design methodology allows different processing levels of intelligence and abstraction. High levels perform planning, supervision and sensor fusion activities. Low levels are charged with actuators control, sensor's information gathering and reactive actions execution. We have developed a software architecture based on *"behaviour-oriented commands"* (BOC). A BOC is carried out by a group of co-operating behaviours which can execute at the same time.

## 3. BOCs -Behaviour Oriented Commands

Following the notion of the agent proposed by Minsky [MIN85], a single agent can be individually seen as a simple process which activates and inhibits other agents and which can itself be activated or inhibited. From an external point of view, an agent is seen as a complete service resulting from the co-operative actions performed by the agent and by an optional set of associated agents.

A behaviour has been defined by Mataric [MAT94], as a control law that clusters a set of constraints in order to achieve and maintain a goal. Our notion of behaviour integrates the two previous concepts. A behaviour can be seen as an autonomous agent that co-operates with others to carry out and/or to respect the rules that yield to a goal.

Simmons [SIM94] proposed the need of using two types of behaviours, deliberative and reactive ones, to achieve a coherent global action of a mobile robot evolving in a real unknown environment. A deliberative behaviour acts in order to implement a set of actions leading to a specific goal, the plan can be modified dynamically as a function of the stimulus received during its execution. In opposition a reactive behaviour acts in answer to a stimulus without taking into account a global plan but respecting a coherent way of acting. In general, stimulus activates and/or modifies the actions of the behaviours. A stimulus is a discrete and asynchronous event, which can also have some information associated to it.

Once an event arrives or an action is carried out a fact is generated. This fact triggers a rule or a set of rules that will produce the next action of the plan to be executed. These actions are direct physical interactions or complex services needed to react properly to the generated fact. Each service is analogue to a command in that it must be requested, executed and acquitted. To carry out a command, one or several behaviours co-operate simultaneously in order to solve problems beyond the scope of each one independently.

Our "behaviour oriented commands" BOCs are the result of fusioning the notions of service and co-operative behaviours. The system is analogue to a dynamic automata network where each automata can perform in parallel. A BOC is defined as a control unit and a set of associated connected behaviours that carry out a service/action respecting a specific set of rules. Figure 1 illustrates the composition and interactions of a BOC. Each behaviour can be seen as an automata whose actions are either executed internally or requested to other behaviours by BOCs.

The control unit associated to a BOC is composed of a BOC's interface, a behaviour's interface and a set of control rules. The former one includes an entry port for the BOC's request and an output port for the BOC's acknowledgement. The second one includes the control links between the BOC and its *"command associated behaviours"* (CABs). The links from the BOC to the CABs are of two types: activation signals and inhibition signals. End signals are sent from the CABs to their associated BOCs. The last one warrants the co-ordination and synchronisation of the CABs by the use of a set of activation, ending and inhibition rules, which can be formalised by a finite state machine or a Petri net.

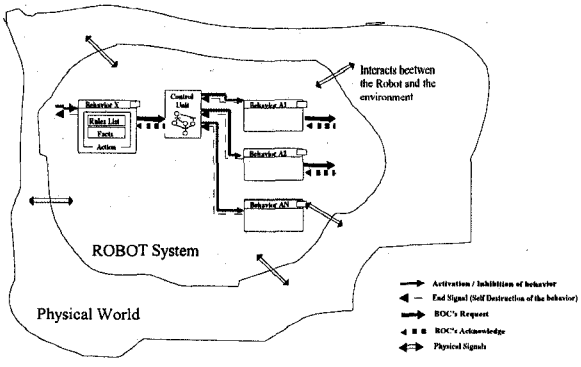


Figure 1. Schematic representation of a BOC

The normal sequence of treatment of a BOC request is:

- Once an event arrives to a behaviour, its fact base is modified, triggering a rule that will find the next BOC to execute.
- The concerned behaviour demands the execution of the required BOC by sending a request.
- The BOC request is received by its control unit and it actives its associated CABs.
- All the CABs start working independently in parallel to achieve their desired behaviour.
- An end signal is generate when one or some of CABs detect a BOCs ending condition.
- When the end signal arrives to the BOCs control unit, it sends an inhibition signals to stop all its CABs.
- The control unit calculates and sends the BOCs acknowledgement to the demanding behaviour.
- This acknowledge includes output data and information about the end of the BOCs execution. This information can modify the behaviour's base of facts and thus a new similar sequence is started.

The end signal can be produced by a behaviour activated by another control unit, the activation/inhibition process is dynamic and the hierarchy might evolve in time.

## 3. Example of Mobile Robot behaviours decomposition:

A mobile robot application is presented. Only the active behaviours at time T are shown. We use the top-down model to decompose our system in term of behaviour network. To begin this decomposition we presume the existence of a " Life behaviour ". Like other behaviours it is composed of its own rule list and state variables which together can infer a BOC request from its BOCs list.

Life Behaviours state variables :

| |
|---|
| BUTTON_SIGNAL= { ON , OFF} |
| ROBOT_MISSION = { EMPTY , DOING , FINISH} |

Life Behaviours BOCs List :

| |
|---|
| ACTIVATE_MISSION( arg1) |
| arg1= ( ACQUISITION_TERAIN , BIBERONAGE) |
| return ={ BOC_OK, BOC_KILL , BOC_MISSION_FAILLED, BOC_LOW_BATTERY , BOC_CONTACT } |

Life Behaviours rule list :

| |
|---|
| If BUTTON_SIGNAL is ON and ROBOT_MISSION is EMPTY then Call ACTIVE_MISSION(ACQUISITION_TERRAIN) BOC and ROBOT_MISSION is DOING. |
| If ACTIVATE_MISSION BOC(ACQUISITION_TERRAIN) acknowledge is BOC_OK then ROBOT_MISSION is now FINISH |
| If BUTTON_SIGNAL is OFF and ROBOT_MISSION is DOING then Kill ACTIVE_MISSION BOC |
| If ACTIVATE_MISSION BOC acknowledge is BOC_KILL then ROBOT_MISSION is EMPTY |
| If ACTIVATE_MISSION BOC acknowledge is BOC_MISSION_FAILED then ROBOT_MISSION is EMPTY |
| If ACTIVATE_MISSION BOC acknowledge is LOW_BATTERY then Kill ACTIVATE_MISSION |
| If LOW_BATTERY then Call ROBOT_MISSION ( BIBERONNAGE) and ROBOT_MISSION is DOING |
| If ROBOT_MISSION ( BIBERONNAGE) acknowledge is BOC_OK then ROBOT_MISSION is now FINISH and call ROBOT_MISSION(ACQUISITION_TERRAIN) |

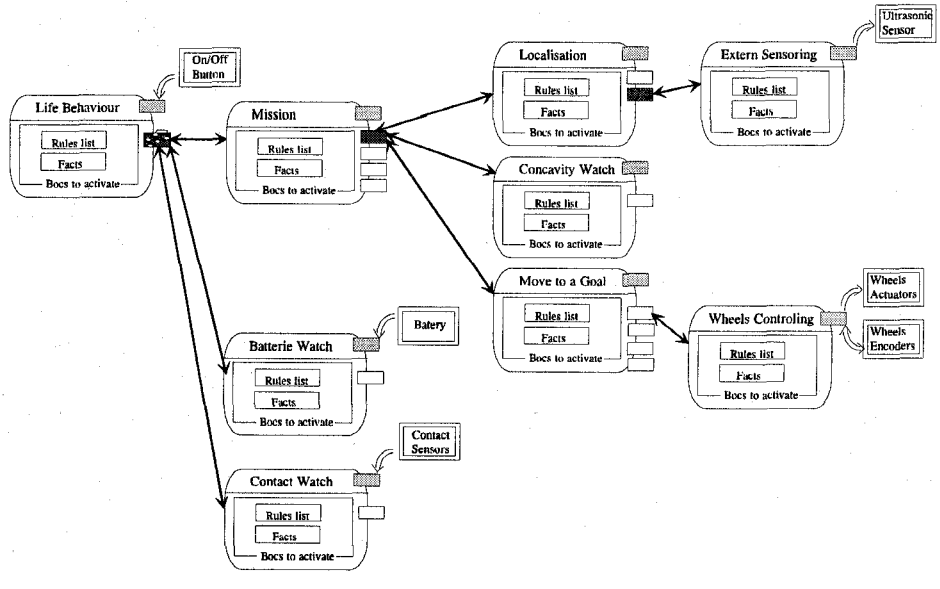


Fig. 2. Active Behaviours at time T in a Mobil Robot application

The example shows active behaviours at instant T. More details or others behaviours are not shown because of size limitations. If a modification of a behaviour is wanted one has only to change its properties (rules list, state variables list or BOCs list).

## 4. Real Time Implementation

A modular and incremental behaviour design based on the previous approach has given life to our robot. We started by implementing the low level BOCs like Explore, Move, Stop_When_Contact, Avoid_Collision. Then the high level ones, like Fill_Region, Circumnavigate_Obstacle, Move_Straight, Explore_World, have been built.

Situation parallelism is naturally introduced by behaviours simultaneity. It is implemented by a distributed software design over a parallel hardware architecture. The computing resources involved in knowledge treatment are distributed among big granularity processes (real time tasks) under temporal constraints.

Co-ordination is achieved by means of real time communication and synchronisation mechanisms. State changes in physical systems or in environment perception must be taken into account within time intervals assuring correct reaction of the autonomous robot. Spatial distribution among elements implies a distribution of the control structure of our system.

### BOCs and Behaviours Implementation

The BOCs architecture which allow parallelism and co-operation between behaviours, involve the use of a set of control tools to guarantee the global coherence of remanent data describing system's state. These tools are introduced to control the global coherence and have to avoid overloading system representation. In order to, they are represented in our model in the form of graphical objets that can be automatically converted into real time primitives by a graph compilation mechanism. during the implementation phases.

For example each behaviour is represented by a rounded square during the design phase and is assimilate to a real time task when implemented. Each BOC's unit control is represented by a rectangle with arrow links between its CABs. In that case too, each control unit is implement as an independent task which communicates with their own CABs by messages or ADA's like rendez-vous.

To guarantee a concurrent access to a shared resource, we use mutual exclusion mechanisms. Each critical resource is stamped to be write/read protected.

We show in figure 3 and figure 4 the global sketch of a task implementing a generic behaviour and a BOC control unit respectively. We use a C language to represent the body of each task. Real time primitives are explicit C functions as found in usual real time kernels.

```
void  Generic_Boc_Control_Unit(void)
{
  msg_type   *msg_data, *end_cab, *boc_request,
  *boc_ack;
  stack_type   END_CABs_LIST;
  int ack_miss, i;
  while(1)
  {
    //accept BOC request
    Receive_Msg (&boc_request , INFINITE_TIME);
    //CABs activation
    for (i=1 ; i < boc_request->
    nb_cabs_to_activate ;i++)
    {
      msgdata=Make_Msg_CAB( boc_request ,
      ACTIVATION);
      Send_Message ( boc_request->
      id_cab_to_activate[i] , msgdata);
    }
    //wait end signal fin from CaB
    ack_miss=boc_Request->nb_cabs_to_activate;
    Receive_Message(&end_cab, INFINITE_TIME)
    ack_miss--;
    putlist( END_CABs_LIST, end_cab);
    //Create inhibition links with the
    N-1 active CABs
    for (i=1 ; i < boc_request->
    nb_cabs_to_activate;i++)
    {
        msgdata=Make_Msg_CAB( boc_request ,
        INIBITION);
        Send_Message ( boc_request->
        id_cab_to_activate[i] , msg_data);
    }
    while ( ack_miss)
    {
      //wait for CABs return information
      if (Receive_Message( &end_cab,
      mbox_cabs,NULL_TIME))
      {
         putlist( END_CABs_LIST, end_cab);
         ack_miss--;
      }
    }
    //build BOC's acknowledge
    boc_ack=Make_Boc_Ack( END_CABs_LIST);
    //Send acknowledge to father Behaviour
    Send_Message( boc_request->
    id_behaviour_father, boc_ack);
  }
}
```

Fig. 3 Generic BOC control unit

```
void  Generic_Behaviour(void)
{
  req_type *boc_request;
  ack_type *boc_ack;
  cab_msg_type *activation_cab;
  list_type   END_CABs_LIST=NULL;
  int ack_miss, i, inibition;

  //Initialisation of facts list
  Init_Facts_Base(behaviour_base);

  while(1)
  {
    //wait for activation
    Receive_Message (&activation_cab ,
    INFINITE_TIME);
    inibition=FALSE;

    //wait for non blocking inhibition
    while(! inibition)
    {
      //chose BOC to execute and ask for
      request
      boc_request=Find_Boc(behaviour_base);
      Send_Message(boc_request->
      unit_control,boc_request);

      //wait ack_request
      Receive_Message (&boc_ack , NULL_TIME);

      //change facts list
      Update_Facts_Base(boc_ack,
      behaviour_base);

      //look for events
      if (Receive_Message(&inibition_cab ,
      NULL_TIME))
      {
        inibition=TRUE;
      }
    }
  }
}
```

Fig. 4 Generic behaviour

Each behaviour waits for an activation signal while each control unit waits for a BOC's request message. When a behaviour is activated, it verifies if there has been an inhibition signal before doing a BOC's request. A behaviour makes a BOC's request and waits for its acknowledgement. The request is made by sending a message to the process that performs the BOCs control unit. If there are several simultaneous request to the same BOC, they are treated in a priority FIFO way. Once the unit control receives the request, it activates its CABs by sending a message to them, and then it waits for an end signal. When a behaviour detects the end condition, it produces this end signal and the unit control sends inhibition signals to all its CABs. Finally it builds and sends the BOCs acknowledge message, and waits for a new request.

## Process Management on a Processor

An asynchronous approach has been used to develop our system, as it is well adapted to specification and conception of applications where tasks are strongly parallel and co-operative. Temporal constraint verification and task co-ordination (scheduling and synchronisation) have been carried out by employing a pre-emptive real time kernel (light process parallelism). A dynamic priority mechanism is used to guarantee system reactivity to urgent non-synchronous events. In order to avoid deadlock problems caused by priority inversion, co-operating tasks have the same priority. They work in a time slicing mode, thus allowing a situation parallelism treatment. Communication and synchronisation between process embedded in the same processor is performed using synchronous messaging (ADA's rendez-vous type-like) and non-synchronous messaging (mailbox mechanism type-like) [LAP93].

## Hardware Architecture and Communications

We have chosen a parallel physical architecture (fig. 5), with processors communicating by pairs allowing a large amount of data exchange. This topology is well adapted to embedded applications requiring modules with strong co-operation like ours. A request and acknowledge protocol is utilised to communicate and synchronise the processes. Our application is composed of a great number of them and they are physically distributed over several hierarchically organised processors.

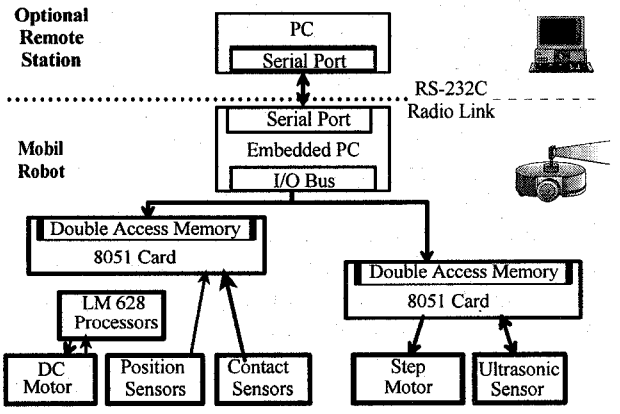


Fig. 5. Hardware Architecture

Communication between the embedded PC and the remote control one is accomplished by a radio serial link. Communication's temporal determinism is guarantied by the use of a protocol with frame error handling. Communication between the embedded PC and the two 8051 is performed by a double-access memory with non-synchronous communication messaging type-like protocol. Request and acknowledge memory zones are dynamically allocated and released. Double access memories are handled like critical resources and protected by mutual exclusion mechanisms.

Processes communicate with each other by using messages managed by the real time kernel (communication inside a processor) and/or by a communication manager (communication between processors). This allows each one to send messages without taking into account the physical location of the destination one. Hence transparency to topological network architecture is obtained.

## Experimental Results and Conclusion

Using this approach a mobile robot has been built and successfully tested. The robot evolves through the environment in a continuos fashion performing simultaneously its mission strategy, localisation, exploration, cartography, control motion and security activities. Therefore validating the correctness of our real time architecture. A high level of intelligence has been reached with low hardware requirements. Our distributed real time architecture and it's convergence to knowledge management systems allows successful implementation of conceptual representation of the robot's tasks, formal behaviour rules and mechanisms improving the reactivity to events.

The model developed allows to manage, to reuse and to modify the robot's knowledge in an non complex way as it incorporates and locates clearly the semantic rules. The DAI approach proposed and its conjunction with the real time systems might be used in other types of complex systems. Integration of two aspects, a distributed real time architecture and a dynamic capitalisation of the knowledge, opens the way to future developments.

## References


[BRO86] Brooks R. A., «A Robust Layered Control System for a Mobile Robot», *IEEE Journal of Robotics and Automation,* Vol. 2, No. 1, 1986, 4 pp 535-539.

[BRU94] Brunet E., Ermine J.L., «Problématique de la Gestion des Connaissances des Organisations», *Revue d'Ingénierie des Systèmes d'Information,* Vol. 2, No. 3, 1994, pp 263-291.

[DEC87] Decker.K.S., «Distribued Problem Solving Techniques: A survey», *IEEE Trans on Systèms, man and cybernetics, Vol 5 No 17,Sept-Oct* 1987.

[ELF86] Elfes A., «A Distributed Control Architecture fon an Autonomous Mobile Robot», *Artificial Intelligence, Vol. 1, No. 2, 1986.*

[GEO87] Georgeff M.P., «Planning»,. *Annual Review of Computer Science*, Vol. 2, 1987, pp. 359-400.

[LAP 93] Laplante. P. A., «Real time systems desing and analysis», *IEEE Computer Society Press, 1993.*

[MAT94] Mataric M.J., «Interaction and Intelligent Behavior», *PhD Thesis MIT,* 1994.

[MIN85] Minsky M., «The Society of Mind», Ed. Simon and Schuster New York, 1985.

[SIM94] Simmons R., «Structured Control for Autonomous Robots», *IEEE Trans. on Robotics and Automation,* Vol. 10, No. 1, Feb. 1994.

[SUA95] Suárez A., González E., Cabo J.C., Rollot Y., Manuel B., Moreno C. Artigue, «An Autonomous Vehicle for Surface Filling», *Proc.IEEE Intelligent Vehicles '95 Symp.*, Sep. 1995.