A Distributed Architecture for Enabling Autonomous Underwater Intervention Missions

N. Palomeras, J. C. García, M. Prats, J. J. Fernández, P. J. Sanz, P. Ridao
University of Girona & University Jaume I, Spain

Abstract— **This work introduces the main aspects related with a new architecture defined for an ongoing research project named RAUVI (*i.e. Reconfigurable AUV for Intervention Missions*). Two initially independent architectures for the underwater vehicle and the robotic arm have been combined into a new schema that allows for reactive and deliberative behaviours on both subsystems. Reactive actions are performed through a low-level control layer in communication with the robot hardware via an abstraction interface. On the other hand, the intervention mission is supervised at a high-level by a Mission Control System (MCS), implemented using the Petri net formalism. Both, the arm and vehicle perception and control modules communicate with the MCS by means of *actions* and *events*. They also share a centralized database where some sensor data is stored. The proposed architecture allows for the supervised execution of intervention missions requiring a tight coordination between the vehicle and the manipulator.**

Keywords— Distributed Architectures, Behaviour Robotics, Autonomous Underwater Vehicle for Intervention (I-AUV).

I. INTRODUCTION

Nowadays, many important applications of underwater robotics in different fields such as marine salvage, marine science and offshore industry, do not only need exploration capabilities, but also intervention skills. While traditionally, intervention tasks were constrained to remotely operated vehicles provided with manipulators, very recently, the introduction of the so-called I-AUV (i.e. Intervention AUV), opened the door to perform intervention tasks autonomously. Without the need for the umbilical cable, I-AUVs can be operated from ships of opportunity, not needing dynamic positioning capabilities and hence, reducing drastically the operational cost. I-AUVs have the potential to become an enabling technology opening the door to new exciting applications like permanent AUV deployment as well as making economically viable others like the maintenance of permanent observatories. However, this new technology poses new challenging problems to the scientific community like the join control of the vehicle-manipulator system or, from the systems engineering point of view, how to design and implement the whole system architecture. The design of the architecture for an autonomous vehicle or a manipulator alone has been a topic of interest during the last twenty years. However, few works have considered the tight cooperation of both systems together in order to perform a join mission autonomously. This paper presents our design to integrate two control architectures initially designed and implemented independently, one to control an AUV and the other to control an underwater manipulator. System's integration is achieved by means of a Mission Control System (MCS) allowing the user to program the mission, uploading it into the I-AUV as well as launching its real-time execution. The MCS is the part of a system's architecture that is in charge of coordinating the high-level phases to be carried out by the vehicle and the manipulator in order to fulfill a predefined mission. Each high-level mission phase is denominated as a task that can be executed by means of enabling/disabling some vehicle/manipulator primitives. The MCS defines how the mission is divided into a set of tasks and how primitives are combined to fulfill each task. The development of a MCS for an autonomous system lies at the intersection between a Discrete Event System (DES) responsible for enabling and disabling basic primitives when some events are produced and the Continuous State Dynamic Control System (CSDCS) used for every primitive to achieve a specific goal. The DES must ensure the consistency of the resulting controller avoiding driving the autonomous system into a deadlock situation and simultaneously ensuring the reachability of one of the final states described in the mission plan. Our approach to the MCS is based on the Petri net formalism for the DES representation to model, program and execute missions. When connecting the two independent architectures, additionally to the MCS, a centralized blackboard has been added to connect both systems. Through the blackboard, it is possible to interchange relevant information between the architectures in order to execute their primitives in a more efficient way while keeping the distributed nature of the system's design.

This paper is organized as follows: In Section II the state of the art of I-AUVs is presented, section III defines the proposed architecture for the integrated I-AUV system and an example of a typical intervention mission is discussed in Section IV before the conclusions.

II. STATE-OF-THE-ART

Pioneer projects using a 6 DOF AUV with a 1 DOF manipulator were the ODIN AUV [1] (University of Hawaii), the OTTER AUV (MBARI) [2]. More evolved, the UNION European project equipped the 5 DOF VORTEX ROV (operated as an AUV) with a 7 DOF Mitsubishi PA10 manipulator [3]. Used as research test-beds, these vehicles were used to prove concepts as the advanced hydrodynamics modeling of underwater arms or the coupled AUV-manipulator simulation and control, being at most operated in water tank conditions.

During the mid 90s, the AMADEUS EU project [4] supposed an improvement of the dexterity and sensory abilities of underwater manipulation systems by means of the design of a new multi-fingered gripper, with tactile and force feedback sensing and by the use of two 7-DOF ANSALDO manipulators in cooperative mode. In 2001, the SWIMMER project [5] developed an autonomous shuttle carrying a work-class ROV

978-1-4244-5883-7/10/$26.00 ©2010 IEEE

to a subsea installation. After autonomously docking into the seafloor docking station, the ROV was remotely deployed and controlled from the seafloor infrastructure without the need for a support ship and avoiding long umbilical cables. An step forward in autonomous intervention was achieved in the ALIVE (EU) project [6] which proved that a 4 DOF I-AUV can automatically identify an unmodified subsea structure, transit to a point nearby and reliably dock to it using two hydraulic grabbing bars. Once grabbed, a 7 DOF manipulator was used to perform simple tasks like opening a valve assuming the panel was a priori known. Probably the most advanced I-AUV nowadays is the SAUVIM (USA) project [7], which is an AUV with a 7 DOF electrical driven arm (ANSALDO), which is intended to recover objects from the seafloor using dexterous manipulation. The real-time architecture of SAUVIM [8] is based on a division between ''low-level'' control (vehicle control) and ''high-level'' control (mission control) that uses a similar configuration with a precise role separation between them.

III. THE PROPOSED ARCHITECTURE

The proposed architecture combines the pre-existing manipulator and vehicle system architectures in a single one able to define intervention missions for the join vehicle-manipulator system. As shown in "Fig 1", both architectures are defined using several modules distributed in layers. Low-level modules are used to provide access to the sensors and actuators while high-level modules are used to execute reactive behaviours. The modules in between extract relevant information produced by sensors as well as to keep the behaviours hardware independent. A MCS plays the role of the high-level layer being connected with both architectures by means of actions and events sent through an Architecture Abstraction Layer. Using the MCS, the reactive architectures used in the vehicle and in the manipulator are complemented by a deliberative layer able to take decisions according to a mission plan previously defined.

Next sections detail the architectures used in the manipulator and the AUV as well as the MCS and the centralized blackboard used to communicate both systems.

A. Manipulator architecture

Whereas many software architectures have been proposed in the literature in the area of mobile robotics [9], there are few contributions in the field of manipulators [10][11]. As pointed out by [12], a control architecture for manipulators should provide: (i) the manipulation ability, in charge of acting on the environment, (ii) the sensory ability, capable of obtaining information from the world, (iii) the data processing ability, which processes data of system activity, and (iv) the intelligent behavior ability, capable of modifying system behavior according to external information.

Therefore, software architecture for manipulation must contain sensor information processing modules feeding low-level manipulation controllers. But, in addition, an event-driven layer on top of these controllers is also needed, providing a *behavior* to the robot. The behavior layer should be in charge of selecting the most appropriate low-level controllers according to the state of the task.

Taking these goals into account, our manipulation architecture is structured into two layers: a reactive layer containing perception and action modules, and a deliberative layer (the Mission Control System), shared with the vehicle architecture, where the appropriate manipulation actions are chosen according to the intervention mission. An additional layer called Architecture Abstraction Layer provides unified interfaces for sending control commands (actions) to the manipulator and for reading conditions (events) generated by the Perception layer.

1) Perception module

We consider both external and internal perception. External perception includes, for example, the perception of the force at the wrist or the robot's perception of the objects pose. Internal perceptions include proprioceptive information and the references, conditions and immediate goals of the robot. Some examples are the joint values, the end-effector position, the maximum force that the robot's fingers can support, the condition that compares the current force at the fingers with respect to the maximum allowed, the position in space where the robot has to immediately move its hand, etc. The robot is also aware of *conditions* on most of these perceptions, as for example, whether the force at the fingers is greater than a given value, whether the hand has reached a target reference, etc. These conditions generate events sent to the Mission Control System through the Architecture Abstraction Layer.

2) Action module

The manipulation architecture distinguishes between two different types of actions: (i) control actions, where data coming from perceptions are used as feedback to execute control algorithms in a reactive manner, and (ii) perceptual actions, that update the robot internal perception, as for example, modifying a task reference, a target object, etc. A control action takes as input a subset of perceptions and generates a control vector, which is sent to the manipulator. Sensor-based controllers integrating vision, force and tactile information have been implemented as control action modules into this manipulation architecture.

B. Vehicle architecture

The vehicle architecture has the task of guaranteeing the AUV functionality. From the implementation point of view, the real-time POSIX interface, together with the ACE/TAO CORBA-RT ORB, have been extensively used to develop the architecture as a set of distributed objects with soft real time capabilities. These objects are distributed among the two onboard PCs. The architecture is composed by a base system and a set of objects customized for the desired robot. There are classes providing soft real-time capabilities, this is guaranteeing the period of execution of the periodic tasks such as the controllers or the sensors. Another important part of the base system are the loggers. A logger object is used to log data from sensors, actuators or any other object component. Moreover, all the computers in the network are synchronized and hence, all the data coming from different sensors can be time related. The AUV software system is conceived as a

hybrid control architecture in two senses: 1) merging reactive-deliberative control strategies [13] [14] and 2) merging a DES with a CSDCS [15]. The architecture is divided in three modules: Robot interface module, Perception module and Control module.

1) Robot interface module.

This is the unique module that contains software objects that dialog with the hardware. There are basically two types of components: sensor objects responsible for reading data from sensors and actuator objects responsible for sending commands to the actuators. Sensor objects include drivers for the following sensors: a Doppler Velocity Log (DVL), an Ultra Short Base Line (USBL), an Imaging Sonar, a Motion Reference Unit equipped with a Fiber Optical Gyroscope (MRU-FOG), two cameras, a depth sensor and an echo sounder. There are also objects for the safety sensors like the water leakage detectors, internal temperature and pressure sensors that allow for the monitoring of the conditions within the pressure vessels. One actuator object for every thruster is also included. A virtual version of every component allows to transparently connecting the robot control architecture to a real-time graphical simulator allowing performing hardware in the loop simulations [16].

2) Perception module

This module contains three basic components: the Navigator, the Obstacle Detector and the centralized blackboard. The Navigator object has the goal of estimating the position and velocity of the robot combining the data obtained from the navigation sensors. The Obstacle Detector is used to detect the relative position of the robot with respect to the obstacles detected in the world. The Control module uses these data keeping the behaviours independent of the physical sensors being used. The centralized blackboard is a shared database with the manipulator's architecture, which is used to share the vehicle and manipulator perceptions. This module writes the main vehicle perception (position, velocity, detected obstacles, etc.) into a centralized database using TCP/IP sockets and reads from this database the main perceptions communicated by the manipulator. Primitives in the vehicle's architecture as well as actions in the manipulator can use these perceptions.

3) Control module

The control module receives sensor inputs from the perception module and sends command outputs to the Actuators residing in the Robot Interface Module. Several primitives (also called actions or behaviours in the literature) are defined to perform the survey, the approach and the inspection steps of an intervention mission. Primitives can be enabled and disabled and their parameters can be changed by means of *actions* send by the MCS through the Architecture Abstraction Layer. Also, *events* can be produced by the primitives to announce that a goal has been reached or a failure has been detected within the vehicle architecture.

C. Mission control system

The mission control system is the part of the control architecture in charge of defining the task execution flow to fulfill a mission. Each task can be executed by means of some manipulator action and/or some vehicle primitives. The mission programmer must define how these actions/primitives are executed to fulfill each task and how the tasks are combined to fulfill the whole mission. The MCS was developed as generic as possible and it allows for an easily tailoring to different control architectures. To achieve this goal the proposed MCS presents a clear interface with any particular control architecture based on *actions* and *events*. Between the MCS and the vehicle/manipulator architectures there is an Architecture Abstraction Layer (AAL) that adapts these *actions* and *events* to the corresponding instances of the target architecture. The AAL depends on the control architecture being used allowing the MCS to remain architecture-independent. Tasks are the basic building blocks of our system. They are defined using the Petri net formalism [17] and communicate with the hardware by means of *actions* and e*vents* sent through the AAL. *Actions* are associated to transitions in the Petri net and they are executed when these transitions fire. *Events* communicate conditions detected by the vehicle architecture to the mission Petri net. Every *event* is associated with one or more places. If a specific *event* is triggered by the control architecture, all the related places receive a token. Tasks were designed to ensure the following properties:

- All the tasks share the same interface, this is, they have the same number of input and output places. A typical interface has a *Begin* and an *Abort* places as input and an *Ok* and a *Fail* places as output.
- Tasks must be free of deadlocks.
- The reachability graph of every task Petri net must show that the set of places marked when all the possible transitions have already been fired after being started in the rest state and with a valid input marking, are those places that were marked in the rest state plus a valid combination of output places.

If these three conditions hold, it is possible to compound several tasks using control structure Petri nets (also verifying the above mentioned conditions). The resulting net after the compounding, will also accomplish them. It is worth noting that it is not necessary to span the whole reachability tree of the compounded Petri net to ensure the deadlock free as well as the state reachability properties. Spanning it, would have a very high computational cost. The Petri net missions implemented according these rules are able to progress from its starting state to its exit state without sticking in a deadlock. It can be probed [18] that the set of this class of Petri nets is closed with respect to the compounding operation. Hence, the result of compounding two Petri nets with these properties is a new Petri net, for which the same properties hold by construction without the need of further verification. Instead of using graphic tools to describe these Petri nets, our approach uses a Mission Control Language (MCL) [19], able to transform a friendly high-level language into a Petri net. A Graphic User Interface (GUI) is used to identify the target, using a set of images previously gathered by the I-AUV, to specify its localization, its grasping points and the task to be performed. The GUI also allows the programming of the MCL intervention mission to be uploaded into the I-AUV for execution.

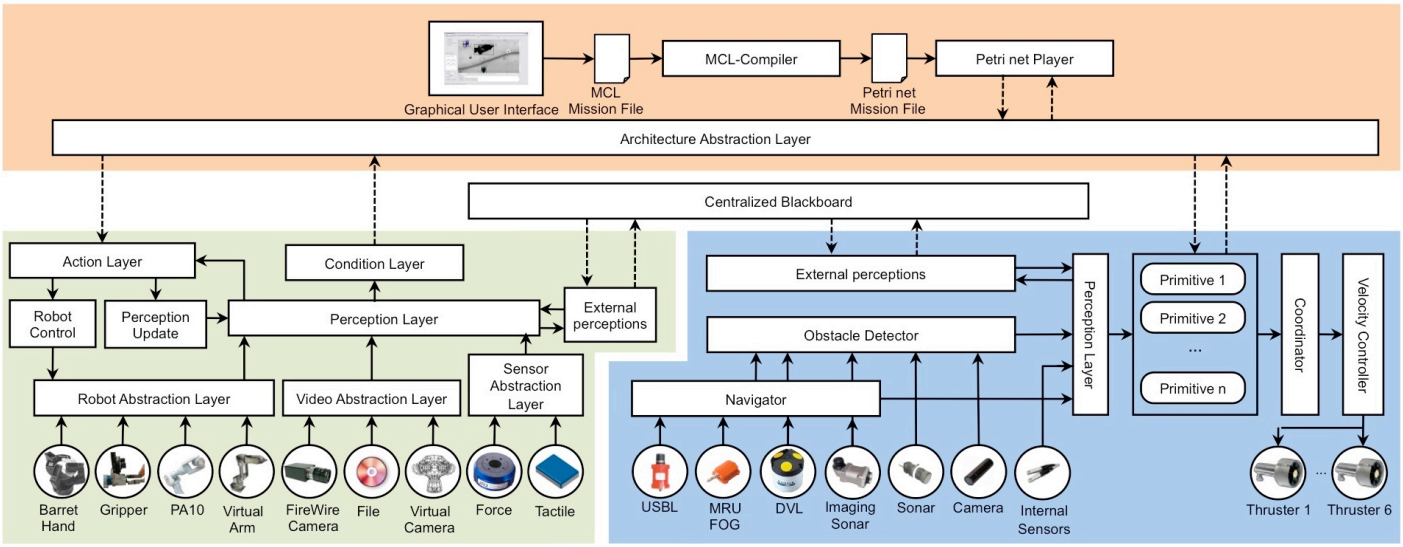

Figure 1**: The proposed architecture**

IV. EXAMPLE OF A MISSION

To carry out a generic intervention mission, we assume that a previous survey is available, probably with multi-beam or side-scan sonar, to identify a reduced area where the target is localized. With this input, the HMI is used to define the area where a visual survey will be executed ("Fig.2"(a)). The HMI output is used to automatically instantiate an MCL survey mission template obtaining the mission program to be downloaded into the I-AUV. After launching the vehicle ("Fig.2"(a) 1.1), the robot follows the programmed survey pattern while recording images synchronized with the navigation data ("Fig.2"(a) 1.2). Once the whole searching area has been covered, the robot is recovered ("Fig.2"(a) 1.3) and the HMI is used again to provide the user with a visual map (image mosaic) of the surveyed area. Over this map, the user selects and characterizes the target, based on visual features, before specifying an intervention task from the library ("Fig.3"). Again, the HMI will be used to automatically instantiate an MCL intervention mission ("Fig.2"(b) 2.1) to be downloaded into the I-AUV before launch ("Fig.2"(b) 2.2). In this second stage, since the target position is known, the robot will approach the Region of Interest (RoI) using its navigation system ("Fig.2"(b) 2.3). Then, with the help of the visual-based navigation the target will be search. At this moment, the I-AUV switch to intervention mode and the vehicle executes a station-keeping task while the manipulator undertakes the intervention ("Fig.2"(b) 2.4). Finally, the vehicle is recovered ("Fig.2"(b) 2.5).

The output of the HMI is an XML script that describes the trajectory to perform during the survey mission as well as the target localization and its visual features characterization in case of an intervention mission (see XML script in "Fig.4").

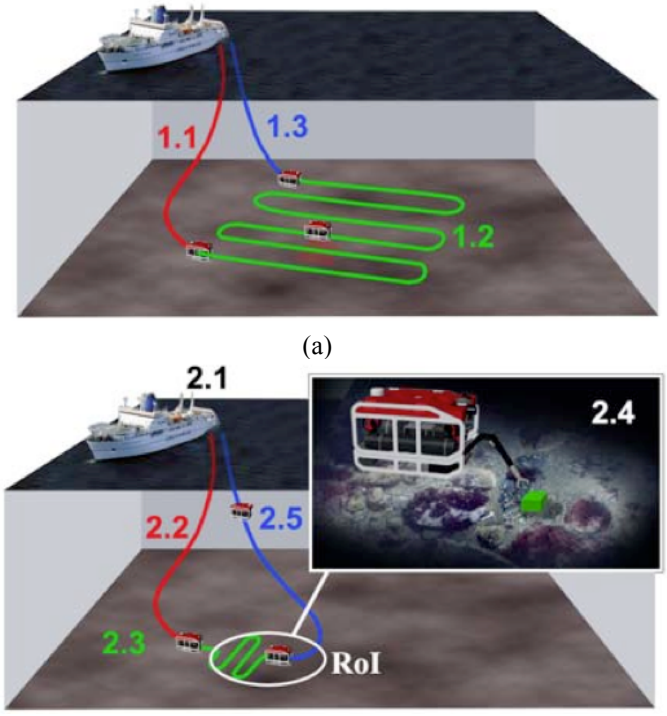

Figure 2: **(a) Survey mission phases. (b) Inspection mission phases.**

"Fig.5" presents the MCL mission template selected by the XML in "Fig.4". The MCL mission combines AUV primitives with manipulator actions, therefore, all the primitives and actions belonging to both systems have to be defined as Petri net tasks within the MCS. When these tasks are defined, it is possible to send actions from the MCS in order to enable these primitives/actions as well as to receive the condition/events produced in both architectures in order to evolve the Petri net tasks.

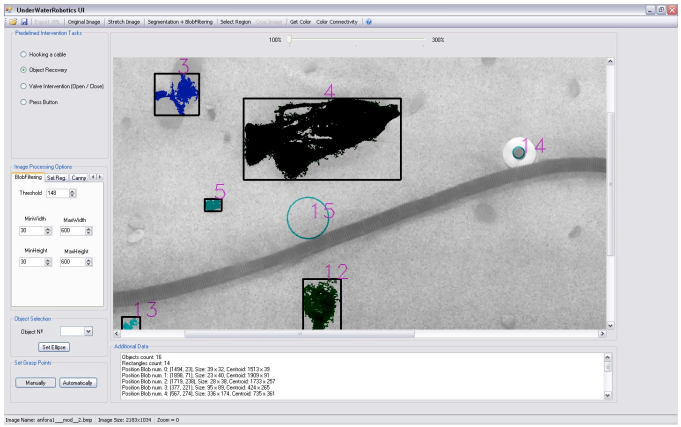

Figure 3: **Example of a target characterization using the HMI.**

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <Intervention>
    <Type>Object recovery</Type>
  </Intervention>
  <Target>
    <TargetWidth>103</TargetWidth>
    <TargetHeight>161</TargetHeight>
    <TargetXCentroid>744</TargetXCentroid>
    <TargetYCentroid>372</TargetYCentroid>
    <TargetOrientation>-15,2551</TargetOrientation>
  </Target>
  <Grasp>
    <GraspLength>76,5310394545899</GraspLength>
    <GraspP1x>715</GraspP1x>
    <GraspP1y>348</GraspP1y>
    <GraspP2x>791</GraspP2x>
    <GraspP2y>357</GraspP2y>
  </Grasp>
</Root>
```

Figure 4: **HMI output after defining an *object recovery* intervention mission.**

When all the tasks are defined, they can be combined using Petri net control structures. In "Fig.5", five Petri net control structures are used, however, other control structures are also available in the language:

- **Sequence**: It is used to execute one task/control-structure after another. It is represented with a semicolon (**;**).
- **Try-Catch-Do**: Executes the Try block in parallel with the Catch block. If the former finishes before the later, the Catch block is cancelled and the execution continues after the Try-Catch-Do structure. However, if the Catch block finishes first, the Try block is aborted and the Do block is executed.
- **Parallel-Or**: Executes two blocks in parallel. The first structure to finish aborts the other. The Parallel-Or finishes with the final state of the first block to end.
- **If-Then-Else**: Executes the block inside the If statement and depending if the block ends with an *ok* or a *fail* the block inside the Then statement or the Else statement is executed respectively.
- **While-do**: Executes the block inside the While statement. If this block finishes with an *ok* executes the do statement, otherwise ends with an *ok*. If the do statement finishes with an *ok* executes again the while statement otherwise ends the whole structure with a *fail*.

Once the mission is coded in MCL, an automatic compiler compounds all the control structure and tasks involved in the mission and generate a single Petri net that can be executed in real-time [19]. This Petri net describes the DES that manages the mission executing the primitives/actions when the appropriate events/conditions are produced.

```
mission {
  try {
    //ENABLE VEHICLE & MANIPULATOR
    Vehicle(ON);
    Manipulator(ON);
    Mode(DEAD_REACKONING);
    Logs(ON);
    parallel {
      //CONFIG VEHICLE NAVIGATION MODE
      while ( True() ) {
        if(HeartBeat(TIMEOUT)) { Mode(USBL) }
        else { Mode(DEAD_REACKONING) }
      }
    }
    or {
      //SEARCH THE TARGET
      AchieveAltitude(SAFE_ALTITUDE);
      parallel { KeepAltitude(SAFE_ALTITUDE) }
      or {
        Goto(OBJECT_X, OBJECT_Y);
        Ligths(ON);
        SearchObject(OBJECT_CHARACTERISTICS);

        parallel{ StationKeeping() }
        or {
          //MANIPULATOR ACTIONS
          ReshapeHand(HAND_CHARACTERISTICS);
          ReachTarget(OBJECT_X, OBJECT_Y);
          while(
            parallel{ LostTargetContact() }
            or {
              Grasp();
              TakeObject()
            }
          ){ ReachTarget(OBJECT_X, OBJECT_Y) }
        }
      }
      //FINALIZE THE MISSION
      Ligths(OFF);
      Goto(RECOVERY_X, RECOVERY_Y);
      Surface();
      Logs(OFF);
      Manipulator(OFF);
      Vehicle(OFF)
  }
} //SAFETY MEASURES
catch {
  parallel { Timeout(MISSION_TIMEOUT) }
  or { LowBattery(MIN_BATTERY_LEVEL) }
  or { WaterLeakage() }
}
do { AbortMission() }
```

Figure 5: **Object Recovery MCL Template**

The MCL template defined in "Fig.5" begins enabling the vehicle and the manipulator as well as their loggers and initializing the vehicle navigation mode. Then the mission starts searching for the target while an independent thread is checking the USBL availability in order to change the navigation mode from DEAD_REACKONING to USBL if the later is available. To search the object, the vehicle is submerged until a desired altitude and then goes to the object position previously defined by the user, turn on the lights and

starts a search procedure to visually localize the object. Once localised, the vehicle keeps its position while the manipulator tries to recollect the specified object. To accomplish this task the hand is reshaped, the object is reached and finally the manipulator grasps the object and takes it. If when grasping or taking the object the contact with the manipulator is lost, the object must be reached again and the procedure restarted. Once the object has been taken the vehicle goes to the recovery area, surfaces and the logs, the manipulator and the vehicle are disabled. In parallel with the whole the mission several alarms are monitored. If the mission exceeds a specified timeout, the batteries fall below a threshold or a water leakage is detected, the mission is aborted.

V. CONCLUSIONS AND FUTURE LINES

The aim of this paper is to introduce the main aspects related with the new architecture defined for an ongoing research project named RAUVI [20] (Reconfigurable AUV for Intervention Missions) involving an AUV and an underwater manipulator. The main focus of this article falls within the description about how the cooperation of both subsystems can be achieved by means of the MCS proposed. An example of an intervention mission is also introduced in order to present an example of how the whole system performs.

ACKNOWLEDGMENT

This research was partly supported by the European Commission's Seventh Framework Programme FP7/2007-2013 under grant agreement 248497(TRIDENT Project), by Ministerio de Ciencia e Innovación (DPI2008-06548-C03), and by Fundació Caixa Castelló-Bancaixa (P1-1B2009-50).

REFERENCES

[1] Choi. S.K. and Yuh. J.. "Design of Advanced Underwater Robotic Vehicle and Graphic Workstation". 1993 IEEE Conference on Robotics and Automation.

[2] H. H. Wang, S. M. Rock, and M. J. Lee. "OTTER: The Design and Development of an Intelligent Underwater Robot". Autonomous Robots, 3(2-3):297-320, June-July 1996.

[3] Rigaud, V.; Coste-Maniere, E.; Aldon, M.J.; Probert, P.; Perrier, M.; Rives, P.; Simon, D.; Lang, D.; Kiener, J.; Casal, A.; Amar, J.; Dauchez, P.; Chantler, M., "UNION: underwater intelligent operation and navigation," Robotics & Automation Magazine, IEEE , vol.5, no.1, pp.25-35, Mar 1998.

[4] Lane D. M., O'Brien D. J., Pickett M., Davies J. B. C., Robinson G. , Jones D., Scott E., Casalino G., Bartolini G., Cannata G. , Ferrara A., Angeletti D., Veruggio G., Bono R., Virgili P., Canals M., Pallas R., Gracia E., Smith C., "AMADEUS-Advanced manipulation for deep underwater sampling", IEEE Robotics and Automation Magazine, pp. 34-45, Vol. 4, No. 4, Dicembre 1997.

[5] Evans, J.C.; Keller, K.M.; Smith, J.S.; Marty, P.; Rigaud, O.V., "Docking techniques and evaluation trials of the SWIMMER AUV: an autonomous deployment AUV for work-class ROVs". OCEANS, 2001. MTS/IEEE Conference and Exhibition, vol.1, no., pp.520-528 vol.1, 2001.

[6] Evans J., Redmond, P., Plakas, C., Hamilton, K and Lane, D, "Autonomous docking for Intervention-AUVs using sonar and video-based real-time 3D pose estimation", OCEANS 2003. Proceedings , vol.4, no., pp. 2201-2210 Vol.4, 22-26 Sept. 2003.

[7] Yuh, J.; Choi, S.K.; Ikehara, C.; Kim, G.H.; McMurty, G.; Ghasemi-Nejhad, M.; Sarkar, N.; Sugihara, K., "Design of a semi-autonomous underwater vehicle for intervention missions (SAUVIM)", Underwater Technology, 1998. Proceedings of the 1998 International Symposium on , vol., no., pp.63-68, 15-17 Apr 1998.

[8] Giacomo Marani, "Advances in Autonomous Underwater Intervention for AUVs", Work Shops and Tutorials, 2009 IEEE International Conference on Robotics and Automation 12-17 May Kobe, Japan.

[9] Bekey, G. 2005. "Autonomous Robots". The MIT Press.

[10] Kröger, T., Finkemeyer, B., Thomas, U., & Wahl, F.M. 2004 (September). "Compliant Motion Programming: The Task Frame Formalism Revisited". In: Mechatronics & Robotics.

[11] Petersson, L., Egerstedt, M., & Christensen, H. 1999 (October). "A Hybrid Control Architecture for Mobile Manipulation". In: IEEE/RSJ International Conference on Intelligent Robots and Systems.

[12] Siciliano, B., & Villani, L. 2000. "Robot force control". Boston, USA: Kluwer Academic Publ.

[13] Ridao, P., Batlle, J., Amat, J., Roberts, G.N. (1999), Recent trends in Control Architectures for Autonomous Underwater Vehicles. International Journal of Science. Vol. 30, pp. 1033-1056.

[14] Valanavis, P.K., Gracanin, D., Matijasevic, M., Kolluru R., and Demetriou G.A., (1997), Control Architectures for Autonomous Underwater Vehicles, IEEE Control Systems Magazine, pp.48-64.

[15] Coste-Maniere, E., Wang, H.H., and Peuch, A., March (1995), Control Architectures: Proceedings US-Portugal Workshop on Undersea Robotics and Intelligent Control, Libon, Portugal, Published by the University of S.W. Lousiana, 1995, pp.54-60.

[16] Ridao, P., Batlle, E., Ribas, D. and Carreras, M. (2004), NEPTUNE: A HIL Simulator for Multiple UUVs. Oceans MTS/IEEE.

[17] Murata, T. (1989). "Petri nets: Properties, analysis and applications". Proceedings of the IEEE, 77(4).

[18] Palomeras, N., Ridao, P., Carreras, M., and Silvestre, C. (2008). "Towards a mission control language for auvs". In 17th IFAC World Congress, 15028 – 15033. The International Federation of Automatic Control, Seoul, Korea.

[19] Palomeras, N., Ridao, P., Carreras, M., and Silvestre, C. (2009). "Using Petri nets to specify and execute missions for Autonomous Underwater Vehicles". The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. October 1115, 2009 St. Louis, USA.

[20] G. De Novi, C. Melchiorri, J. C. García, P. J. Sanz, P. Ridao, G. Oliver, "A New Approach for a Reconfigurable Autonomous Underwater Vehicle for Intervention", IEEE SysCon 2009 —3rd Annual IEEE International Systems Conference, 2009, Vancouver, Canada, March 23–26, 2009.