3 Intelligent Control Architecture

3.1 Architectural Foundations

This Chapter presents the structural and behavioural aspects of the ICA proposed in this research work. It describes architectural foundations key to develop the ICA and presents conceptual principles as to its structure and behaviour. It involves aspects of the control hierarchy (from goals to behaviours) for the above architecture as well as a detailed explanation of knowledge representation and ontological reasoning methodologies to apply artificial intelligence to UMVs

Figure 3.1 shows the architectural concepts of the ICA. In the top of the figure, the system deals with hierarchical mission goals that are achieved by the execution of agent plans (sequence of activities listed as command messages). The planning and matching are intellectual agent activities. The planning of tasks for an agent is performed by each agent by matching internal agent capabilities but also taking into account external capabilities from other agents to carry out different activities. Agents are able to discover the capabilities of each other.

In the bottom of Figure 3.1, the activities can be seen as service processes (execution of services, e.g. navigation, manipulation, vision, etc.). They can have a basic or composite structure. The basic processes are indivisible, whereas the composite processes can be decomposed into other activities. This composition of activities or service processes is called orchestration of services. It plays an important role in the system architecture since it can define different encapsulation levels to execute services. On the other hand, choreography of services deals with the messages exchanges among services that are executed in parallel (collaborative nature). Orchestration and choreography are terms from Service-Oriented Architecture (SOA). Based on the conceptual structure presented in Figure 3.1, a service-oriented agent-based approach is proposed as ICA.

From the robotics viewpoint, missions, goals, planning, matching, and agents correspond to the “deliberation” layer; services, orchestration, and choreography correspond to the “execution” layer; and activities correspond to the “behaviour” layer. For example, in a single-vehicle seabed survey (mission) the main goal is to collect seafloor data from a given exploration area. The agent is an AUV which plans its tasks (diving, path-following, and surface) by means of checking for availability of its capabilities to carry out such tasks. Services for this mission are from navigation, guidance, control, and vision capabilities to carry out activities (also behaviour) such as “dive”, “emerge”, “capture image”, etc.

C.C. Insaurralde, *Intelligent Autonomy for Unmanned Marine Vehicles,*
Studies in Systems, Decision and Control 29, DOI: 10.1007/978-3-319-18778-5_3

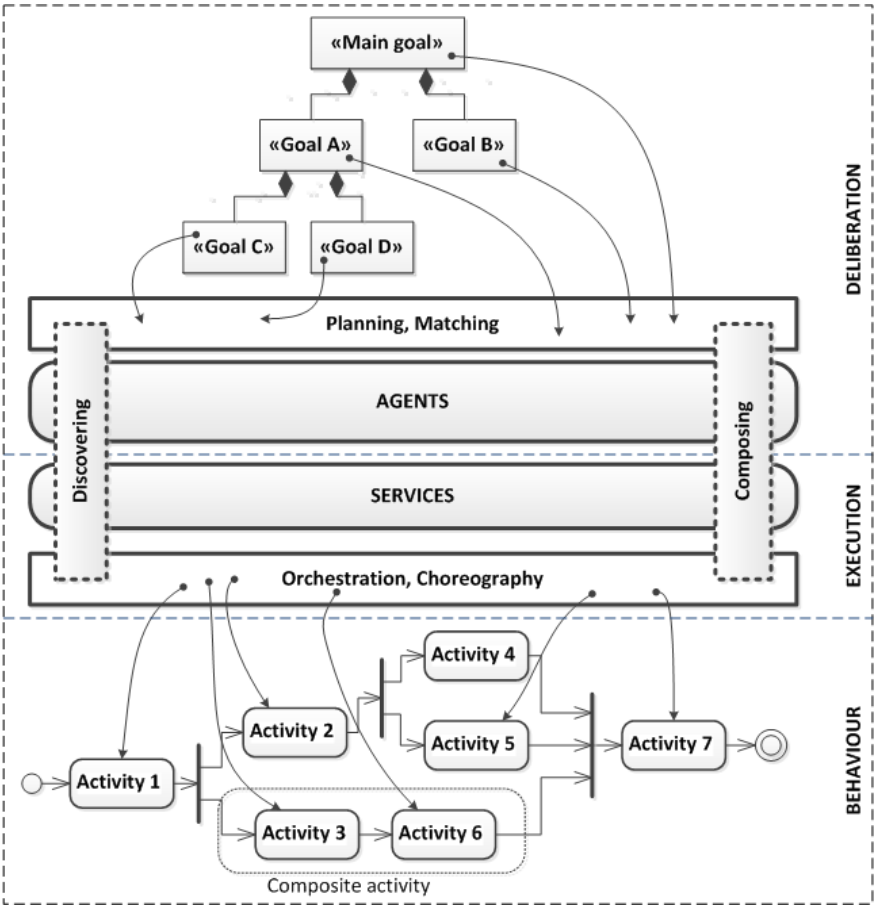


Fig. 3.1 Conceptual view of the service-oriented agent-based architecture

Table 3.1 shows the ICA architectural elements, and their different interaction levels. The activities are classified as mission, operation, task, and action. The services are categorized by following the above activities classification. This hierarchical information classification impacts on the knowledge representation and its design. Ontologies are used to represent the knowledge.

Table 3.1 Interaction levels of the ICA architectural elements

Integration Level	Service	Physical Entities	Logical Entities	Maritime Activities
High	Compositional	Group of vehicles	Holons	Missions
High-mid	Compositional	Vehicles	Agents	Operations
Low-mid	Compositional	Devices	Actors	Tasks
Low	Atomic	Transducers	Workers	Actions

At the lowest level (centre of the Table 3.1), there are actions from transducers (i.e. sensors and actuators). In the next level up, there are tasks from devices that play a role as actors. Above that, there are operations carried out by vehicles which play a role as agents.

At the highest level (top of the Table 3.1), there are missions carried out by group of vehicles that play a role as holons (multi-agent interaction). The basic robotics layers are placed between levels.

Figure 3.2 shows the dependency relations among the key elements of the ICA. The system, i.e. AMRs, fulfils one or more missions (represented by "1..*"), has one or more components (or modules), and use case(s). It also has facilities to sense and act within the environment. Missions are carried out by agents that have one or more goals and plans. A goal is achieved by one or more plans. An agent carries out one or more activities planned according to the platform capabilities, and the goal needs. An activity is carried out by one or more services that encapsulate one or more functionalities of the AMR components. Matching the robotics architecture, functionality means "behaviour".

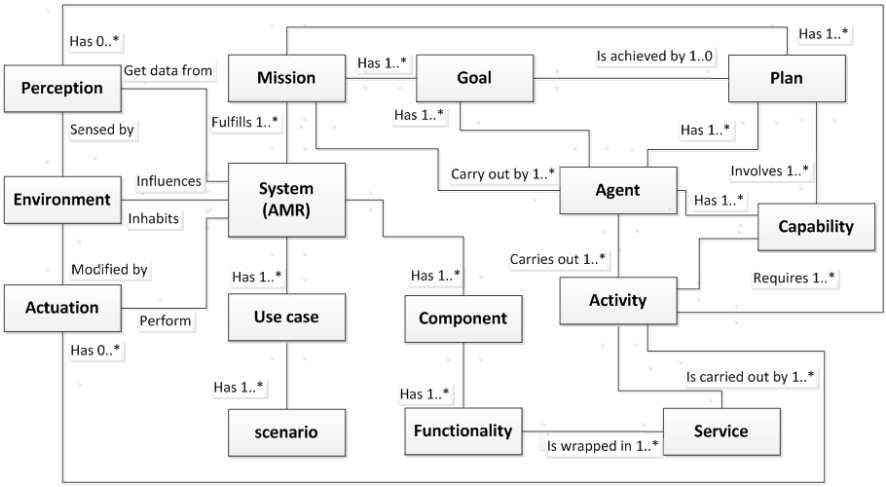


Fig. 3.2 Relationships among the key elements of the ICA

Following the dependency relations presented in Figure 3.2, a bottom-to-top development process for the AMR architecture is defined. It begins identifying the functionalities of the platform components (or modules), and ends determining the plans of the agent to achieve the given mission goals. The development steps are as follows.

Extraction of functionality from platform components (or modules). Grouping and separation of functionalities in order to build clusters of similar functions. Each function can in turn be built of other functions.

Encapsulation of the above functionalities in basic or composite services gives serviceability to the AMR system. It enables the AMR system to carry out activities (service process or execution of services that encapsulate functionalities) at different interaction levels (mission, operation, task, and action). The activities are based on capabilities derived from the component functionalities.

The capabilities are in turn grouped in order to build an agent. The plan of the agent is built according to the mission goals of the AMR. A database stores the knowledge representation of the entire AMR.

3.2 Hierarchical Control

This Subsection presents hierarchical control aspects of the ICA by explaining details of the system architecture integration, AMR hierarchy, and agent anatomy.

3.2.1 System Architecture Integration

Figure 3.3 shows the operation principles of the AMV system. This figure is explained by dividing it into two areas: the top part and bottom part. The former depicts how the system works at the planning level. The latter depicts how the system works at the execution level.

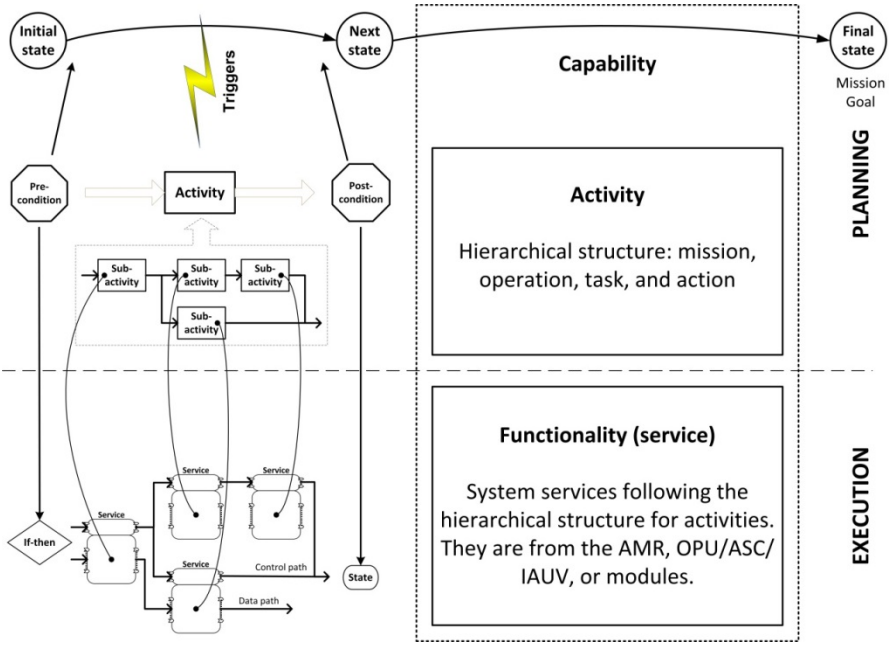


Fig. 3.3 High-low-level agent integration

Figure 3.3 presents the existing connections between the planning and execution levels. The concept shown in this figure can also be applied to the internal operation of an agent, i.e. internal planning and execution of agent tasks in a similar way (strategy) as it is happens in a team of agents.

At the left bottom of Figure 3.3, the network of platform services performs the activities required by the plan (the left top). There is a one-by-one relation between activities and services as shown in the figure. The activities are only triggered when pre-conditions are met. They also generate post-conditions. Pre-conditions are usually evaluated by "if-then" conditional sentences on states of data, and objects. Post-conditions normally result in new states of data, and objects that are used to evaluate the next pre-conditions. Goals are states, so every intermediate state reached can be considered as sub-goals achieved.

On the right of Figure 3.3 is a description of what a capability is, and the two levels it covers. At the right bottom, the network of platform services is the functionality that the system (AMV), subsystems (OCU, ASC, or IAUV), subsystem nodes, and node components provide. At the right top, the activities are hierarchically categorized as missions, operations, tasks, and actions. Thus, a capability is built of activities and functionalities (services).

The capability anatomy is depicted in Figure 3.4. Capabilities are basically divided into two main parts: activity and functionality (service), and are triggered by messages. Activities are carried out by services. Thus, capabilities can be seen from a dynamic viewpoint; activities, and from a static viewpoint; services.

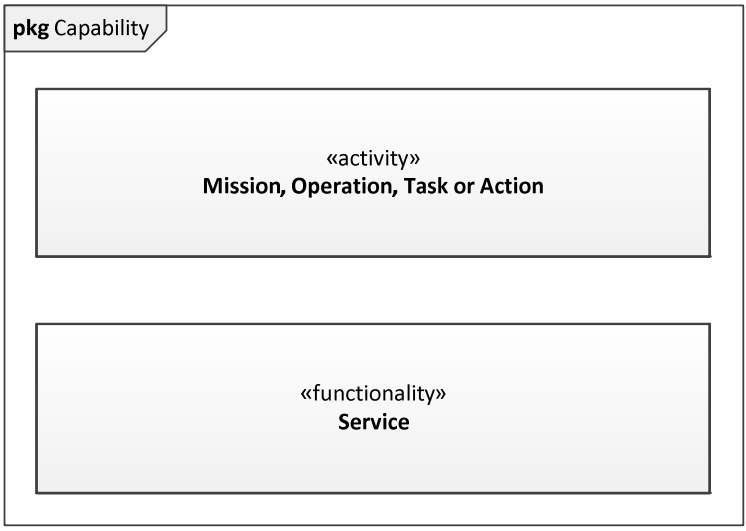


Fig. 3.4 Capability anatomy

As capabilities are built of services, they also follow the classification for services as to hierarchical decomposition as presented in the following Chapter. Therefore, the ICA has two types of capabilities:

- Basic capability: They are indivisible. Therefore, they are the atomic elements of AMR platform in which the ICA is based on.
- Composite capability: They are composed by in other capabilities (basic or composite capabilities).

Following the above classification, capabilities are classified following the hierarchical categorization:

- A mission capability is composed of one or more Operation capabilities
- An operation capability is composed of one or more Task capabilities
- A task capability is composed of one or more Action capabilities
- An action capability are the atomic part of the capability hierarchy

The first three capability classifications are composite capabilities, and the last one is a basic capability. The mission and operation capabilities are designed by wrapping high-level functionalities and operability. The task and action capabilities are designed by wrapping low-level functionalities and operability that provide the platform components.

3.2.2 Autonomous Marine Robot Hierarchy

Advanced computational systems such as multi-agent systems are suitable to implement biological organizations inspired from social behaviour of their members who can be organized in group, community, etc. according to their role in the system. This enables the system to define an organizational hierarchy, and be part and whole of the system at a time.

Holonic structures offer a powerful abstract modelling for large complex systems. An architectural approach to support the above structure in agency (agent community) with collective behaviour exhibited by groups of agents is by means of holonic systems. The main representational concern in this approach is that interacting agents with particular skills behave as if they were a single entity. Based on the holon concept, elementary entity of a holonic system, groups of agents can be organized in a team of coalesced agents. A holon keeps structural self-similarity by being composed of holons as sub-structures. This hierarchical relationship can be extended recursively, and is called holarchy. Thus, a holon can be seen either as an autonomous individual entity or as a hierarchical organization of sub-holons, according to the viewpoint chosen [35].

Figure 3.5 shows the hierarchical multi-agent or holonic system defined for TRIDENT [1]. The OCU agent is at a higher control level where it supervises behaviour of the other two agents (ASC and IAUV). Of course, each agent keeps autonomy all the time but in term of organization, the OCU implements organizational techniques to facilitate the interaction among agents, i.e. communication, coordination, cooperation, and collaboration.

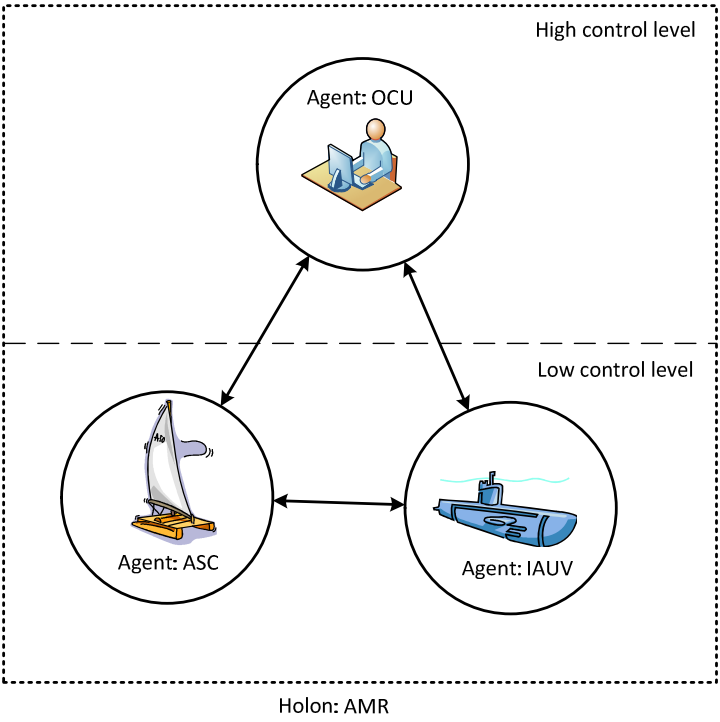


Fig. 3.5 Multi-agent hierarchy

Based on the above holonic structure, the following Subsections describe details of design as to the external behaviour of the AMR agents. They are focused on the mission and operation capabilities provides by the AMR system.

Therefore, planning approach is a global planning for local plan where there are basically two planning: the global plan for the OCU, and the local plans for the ASC and IAUV. They are presented in Section 7.

3.2.3 Foundations for the Agent Structure

The foundations of the ICA have multi-disciplinary nature. It comes from the robotics, cognitive science, and computer science. Therefore, the ICA development is based on the following architectural representations: robotic, cognitive, and agentic models. There are currently different reference models for each of the above representational descriptions. In particular, the ICA combines the following approaches.

A **robotic architecture** which is a hybrid approach composed of three-layer architecture (Planning, Sequencing, and Skill) plus a knowledge block; World Model. Figure 3.6 shows this combined architecture (top left).

A **cognitive architecture** built of two blocks: TBox and ABox which are part of the knowledge representation based on description logics in Figure 3.6 shows the elements of this cognition process (top right).

An **agentic architecture** based on the Belief-Decide-Intention (BDI) software model. The agent structure is shown in Figure 3.6 (bottom). It is built of well-defined blocks, i.e. Belief, Desire (goal), and Intention blocks. Additionally, there are Interpreter and Plan blocks.

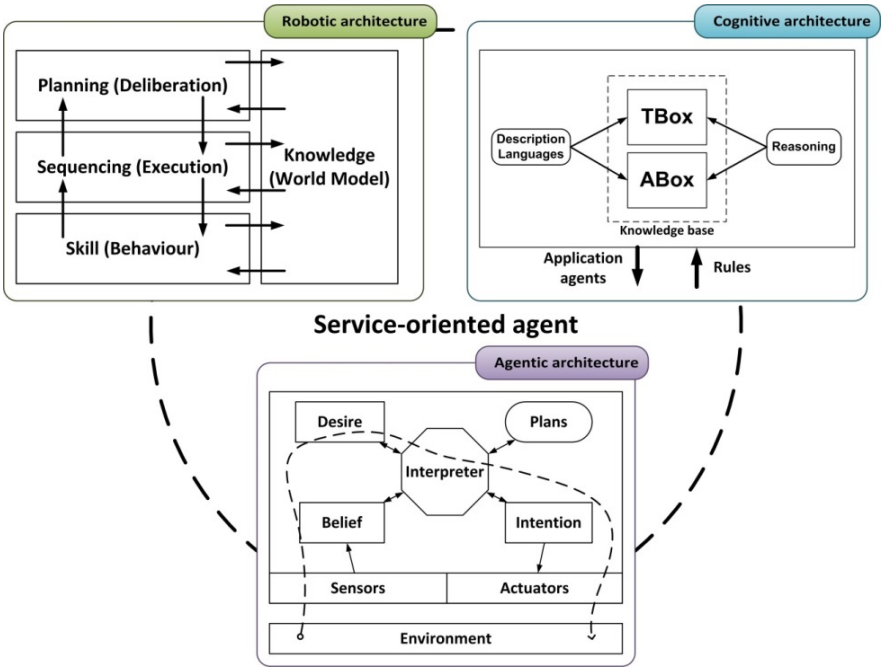


Fig. 3.6 Architectural drivers for the agent structure

Situation Awareness (SA) is the ability to be aware of and understand what is happening in the surroundings of an agent, both at the present time, and in the future through prediction. This capability allows systems to understand dynamic and complex environments, and operate with them. It can be divided into three separate levels: perception of the elements in the environment, comprehension of the current situation, and projection of future status. SA involves the events, states, condition, and activities of the environment dynamics as to time and space from which some situations arise (in particular those changes that occurred in the environment over some time interval). A situation is defined by a specific state after a sequence of events (with intermediate states, and activities with pre and post conditions). The situation is concerned with the comprehension of the environment features, and with the evolvement of these features over time [36].

SA is essential for decision makers. Within an agent, the decision making cycle is defined by four basic stages: Observation-Orientation-Decision-Action (OODA) loop. The Observation stage is the SA perception level. The Orientation stage takes into account the information acquired from the Observation stage and the knowledge of the agent, to understand the situation (SA comprehension level). The Decision stage is carried out at the SA projection level. The Action stage closes the OODA loop by carrying out actions according to the environmental adaption made in the previous stage.

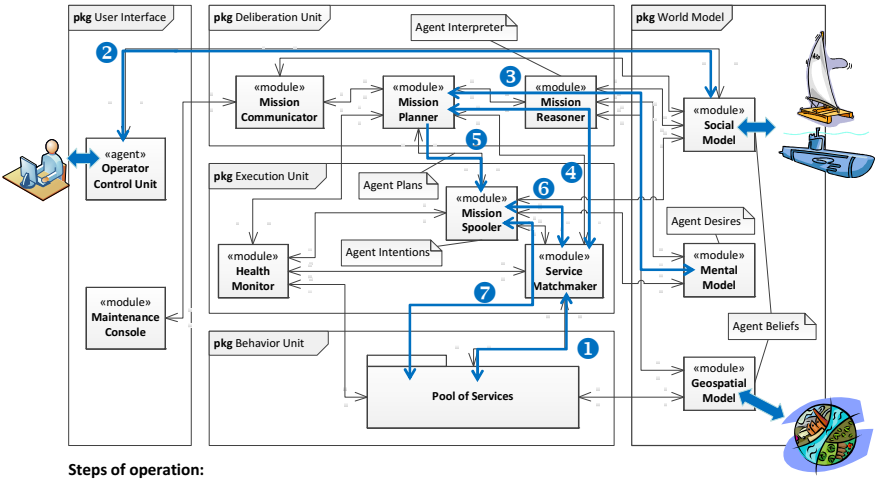
The mapping of the SA and OODA concepts onto the BDI agent architecture is as follows. The Belief block represents the informational state of the agent, and describes the known state of the world (the world model). It matches the SA perception and comprehension levels or OODA observation and orientation stages. The Desire block represents the motivational state of the agent (goals or situations that the agent would like to accomplish). The Intention block represents the deliberative state of the agent (what the agent has chosen to do). It corresponds to the SA projection level or OODA decision and action stages. The Interpreter block maps to the agent reasoner.

The above approach endows the agent with initiative. Decision making mechanisms are critical for problem-solving processes that are preformed every time an agent receives a mission to be carried out.

3.2.4 Agent Anatomy

The agent anatomy is depicted in Figure 3.7. It shows the internal structure of the service-oriented agent. There is one agent per marine vehicle. This figure encompasses three architectural models mentioned above:

- A block-layered robotic model as shown in the centre of Figure 3.7 (linked to the model shown in the top-left of Figure 3.6) with the following blocks: units of planning (deliberation), sequencing (execution), skill (behaviour), and a world model. In addition, a user interface is taken into account.
- A description-logics model as shown in the top-left of Figure 3.7 (linked to the model shown in the top-right of Figure 3.6) which involves the following blocks: deliberation unit (mission reasoner), and world model (mental model; ontology).
- A model with the logical structure of BDI agents as shown in Figure 3.7 (linked to the model shown in the bottom-centre of Figure 3.6): beliefs, desires, interpreter, intentions, and plans.



Steps of operation:

❶ Services are advertised and discovery by means of the service matchmaker.
❷ The operator sets the mission, and communicates it to the team of agents (AMVs).
❸ Each agent (AMV) queries itself in order to know how to deal with the given mission.
❹ Capabilities required by the mission are checking for availability.
❺ The planner sends the mission plan to the mission spooler for execution.
❻ The mission spooler checks status of services though the service matchmaker.
❼ The mission spooler executes task as planned by invoking services.

Fig. 3.7 Agent anatomy

The five main blocks (identified as SysML packages, i.e. "pkg") in the agent anatomy shown in Figure 3.7 are:

User Interface. The end user is able to deal with the mission, and visualize the mission results through an Operator Control Unit (OCU), e.g. seabed map (image mosaicking), scene and objects characterization, etc.

Deliberation Unit. This has basically three components: the mission communicator, the mission planner and the mission reasoner. The mission communicator, which includes the communication manager (wired and wireless communication channels), communicates with the human operator and with the marine vehicles through the social model. The mission planner, which includes the resource manager, helps to selects the agent capabilities required to take actions according to the decisions made by the agent interpreter. The mission reasoner, which includes the agent interpreter, reads the data perceived from sensors, interprets them according to the knowledge embedded in the mental model, and makes the decision of what to do next. The mission planner output is a plan (list of activities to be carried out by the spooler).

Execution Unit. This is in charge of dealing with the execution of the agent services. The execution is according to the plan generated in the mission planner, and it is executed by the mission spooler. The mission spooler is responsible for parceling out activities listed in the mission plan for execution by platform services.

The health monitor deals with the status of the platform services by keeping record of the vital working conditions. It implements the fault diagnosis techniques.

Behaviour Unit. The pool of services of the agent depends on the marine vehicle it is deployed on. They are services at the vehicle level. In the case of the ASC the services provided are: navigation, behaviour management, waypoint list setting, and acoustic/radio communication. In the case of the IAUV the services provided are: navigation, path plan setting, maps generation, seabed data collection, scene/object identification, visual docking, manipulation, grasp specification, and acoustic/radio communication.

World Model. This is a central repository built from the following models. (1) Social Model. It describes the social context which the agent inhabits and interacts with. It is built of the agent directory module which includes the service registry. (2) Mental Model. It describes what the agent is able to know about itself. (3) Geospatial Model. This contains environmental data collected by sensors (perception).

The steps of agent operation are as follows:

1. Advertisement and discovery of services
The first operation step is performed just after the subsystem (only marine vehicles; ASC or IAUV) is switched on. The platform services are advertised to the match maker, and so discover by the subsystem and other services in order to know which of them are available or any other status they may have. They are able to create dependencies as needed but they are not executed. They just wait for that order which comes from the mission spooler later on. Thus, the pool of services is in an idle state waiting for execution.

2. Mission selection
The second operation step is performed after the subsystem (only marine vehicles; ASC or IAUV) has taken note of the available services and their status. The end user selects the mission to be carried out through the operator console which then communicates with the marine vehicle in order to let them know about the mission request.

3. Query on the mental model for capabilities required
The third operation step is performed after the subsystem (only marine vehicles; ASC or IAUV) has received the order to carry out the mission given. The mission planner queries the mental model by means of the mission reasoner in order to get the know-how to carry out the mission. The mental model provides to the mission planner with a kind of "recipe" involving (sub)goals, (sub)capabilities; activities plus functionalities, and (sub)conditions.

4. Query on the matchmaker for service availability
The fourth operation step is performed just after the subsystem (only marine vehicles; ASC or IAUV) is switched on. The mission planner checks the plan consistency (based the capabilities required to carry out the mission) against the

record kept by the match maker. The platform services are basically available or not available. When they are available, the mission planner must check their health status in order to know if he can really make use of them. If there is any problem to execute the services or if they are unavailable, the service matchmaker proceeds to find any other capability that can replace the required one. If no capability are available at all, the service matchmaker must decide what to do the plan (if it is still viable or not), and communicates to the rest of the system (AMR).

5. Plan ready for spooling
The fifth operation step is performed after the subsystem (only marine vehicles; ASC or IAUV) has checked the plan consistency. The mission planner builds the plan based on the platform capabilities, and then sends it to the mission spooler for execution.

6. Retrieval of more details about the services
The sixth operation step is performed after the subsystem (only marine vehicles; ASC or IAUV) is ready to implement the mission plan given. The mission spooler retrieves the all the information needed to execute the services from the matchmaker, i.e. based on the service name, the mission spooler makes a query for status, and invocation method for each service in order to create the execution queue.

7. Plan spooling as services execution
The seventh operation step is performed after the subsystem (only marine vehicles; ASC or IAUV) is ready to execute the platform services. The mission spooler invokes the platform services according to the mission plan, and taken into account the health of the services

The health monitor keeps watching over the entire subsystem by detecting potential faults, and notifying them to the spooler as well as the mission planner.

3.2.5 Agent Dependability

Intelligent agents are developed to cope with uncertainty and abnormal situations. A robust marine robot is that can survive and fulfil its missions despite unforeseen contingencies such obstacles, rough seafloors, and even failures. Thus, robotic robustness is the ability to deal with adverse environments, and any failure whilst providing an acceptable serviceability. Therefore, intelligent agents can be more or less dependable based on the above robustness statement.

All the above contingencies are considered as faults. The challenge for intelligent agents is to know how to deal with the faults found whilst autonomous operation. In terms of faults management, two main process stages can be defined: diagnosis and mitigation.

Diagnosis (identification)
- Detection
- Classification (localization or Isolation)
- Analysis

Mitigation (Means)

- Fault prevention
- Fault tolerance
- Fault removal
- Fault prediction

The faults management process affects to the following agent features: autonomy, architecture, and deliberation. Thus, to keep stable autonomy, the architecture has to efficiently deal with the deliberative process in an agent. In particular, with the decision-making capability in order to not only plan the right activities to be performed but besides increase the likelihood of being successful. Addressing this issue, this report version only takes into account the fault tolerance mitigation applied to the agent planning. To deal with the fault management, a fault-tolerant planning performs diagnosis in order to identify the faults, and then mitigate them by some recovery mechanisms.

Faults Diagnosis. The key step of this diagnostic mechanism of faults identification is the first step above defined, i.e. faults detection. The faults detection techniques supported by the agents implemented in the ICA are as follows:

- Watchdog timing.
- Consistency analysis.
- Effectiveness detection.
- Malfunction detection.

Faults Recovery. There are two basic robustness techniques to recover from a plan failure caused by adverse situations [43]:

- **Re-planning.** It consists in developing a new plan from the current system state, and still unresolved goals. Depending on the planning model complexity, re-planning may be significantly time costly. Other system activities are thus generally halted during re-planning.
- **Plan repair.** It may be attempted before re-planning, with the aim of reducing the time lost in re-planning. It uses salvageable parts of the previous failed plan that are executed while the rest of the plan is being repaired. However, if reducing the salvaged plan conflicts with unresolved goals, plan repair is stopped and re-planning is initiated.

Faults Classification (adapted from [44])

- Nature
  - Unintentional (chance)
  - Intentional
- Perception
  - Conceptual
  - Logical
  - physical

- Boundary
  - Internal (inter)
  - External (intra)
  - (supra)
- Origin
  - Development
  - Deployment
  - Sustainment (Operation)
- Occurrence
  - Non-periodic
  - Sporadic
  - Aperiodic
  - Periodic
- Hierarchy
  - Mission
  - Operation
  - Task
  - Action

3.3 Knowledge Representation

This Subsection presents architectural aspects of the knowledge representation as well as details of the ontology defined for the ICA.

3.3.1 Cognitive Conceptualization

The knowledge representation in the ICA utilizes ontologies. The main ontology elements are concepts (classes), properties, instances (individuals), and assertions. A concept represents a set of entities or things within a domain. Properties define either relations between an individual and a value, or between two individuals; called data type properties, and object properties, respectively. Knowledge representation based on description logics has a block called TBox which defines the concepts and properties in a domain in addition to specifying terminological axioms for every atomic concept (Figure 3.8). Axioms are used to constrain the range, and domain of the concepts, e.g. an IAUV is a marine vehicle that has navigation capabilities. A block called ABox contains a finite set of assertions for the classification of individuals, and the properties they have. The combination of the TBox and the ABox forms the knowledge base that can be described with ontologies. Inference over the ontology is provided by a reasoner.

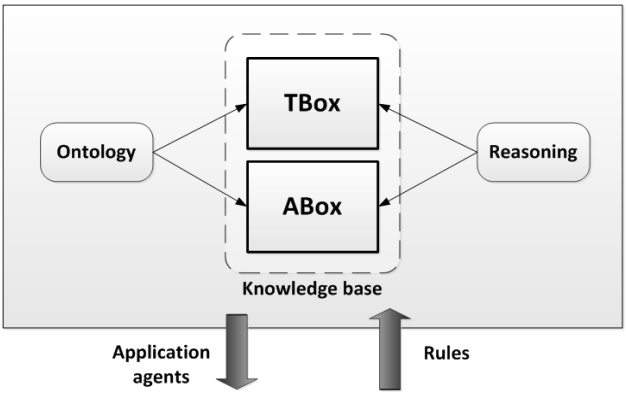


Fig. 3.8 Knowledge representation structure based on description logics

Situation Awareness (SA) is the ability to be aware of and understand what is happening in the surroundings of an agent, both at the present time, and in the future through prediction [23]. This capability allows systems to understand dynamic and complex environments, and operate with them. It can be divided into three separate levels: perception of the elements in the environment, comprehension of the current situation, and projection of future status [24]. The decision making cycle is defined by four basic stages: Observation-Orientation-Decision-Action (OODA) loop [25]. The Observation stage is the SA perception level. The Orientation stage takes into account the information acquired from the Observation stage and the knowledge of the agent, to understand the situation (SA comprehension level). The Decision stage is carried out at the SA projection level. The Action stage closes the OODA loop by carrying out actions according to the environmental adaption made in the previous stage.

The two above concepts, SA and the OODA loop, are the foundation of AMRs. The SA levels for individual unmanned vehicle systems range from fully human controlled to fully autonomous unmanned capabilities [26].

The ICA is based on agents that apply the above SA and OODA concepts. The agent structure selected for the current approach implements a BDI-based architecture. This architecture is built of well-defined blocks, i.e. Belief, Desire (goal), and Intention blocks. Additionally, there are Interpreter and Plan blocks. The mapping of the SA and OODA concepts onto this architecture is as follows. The Belief block represents the informational state of the agent, and describes the known state of the world (the world model). It matches the SA perception and comprehension levels or OODA observation and orientation stages. The Desire block represents the motivational state of the agent (goals or situations that the agent would like to accomplish). The Intention block represents the deliberative state of the agent (what the agent has chosen to do). It corresponds to the SA projection level or OODA decision and action stages. The Interpreter block maps to the agent reasoner.

There are three ontology levels: foundation/upper, core/domain, and application. The ICA only develops core and application ontologies since the foundational (or upper) ontology represents the very basic principles to ensure reusability across different domains.

3.3.2 Foundation Ontology

To lay the foundation for the knowledge representation of unmanned vehicles, consideration was placed on the Joint Architecture for Unmanned Systems (JAUS). This was originally developed for the ground domain only, and has recently been extended to all domains trying to provide a common set of architecture elements and concepts [8].

The JAUS model separates the service-oriented agents, called Functional Agents, in six different functional sets: Command, Telecommunications, Mobility, Payload, Maintenance and Training. It also classifies four different sets of Knowledge Stores: Status, World map, Library and Log. Our experience has shown that an overlap exists between these different sets of knowledge stores. The approach proposed in this book provides more flexibility in the way the information can be accessed and stored, while being JAUS compliant at the communication level between agents.

Core Ontology

Within the proposed framework, JAUS concepts are considered as the foundation for the knowledge representation. The core ontology developed in this work extends these concepts while remaining focused in the domain of unmanned systems.

The knowledge concepts identified as essential parts for maritime systems as vehicles, measurable parameters that are related with this domain are:

Platform: Static or mobile (ground, air, underwater vehicles).
Payload: Hardware with particular properties, sensors or modules.
Module: Software with specific capabilities.
Sensor: A device that receives and responds to a signal or stimulus.
Driver: Module for interaction with a specific sensor/effector.
Waypoint: Position in space with coordinate and tolerance.
Coordinate: Local frame, global frame, angular.
Velocity: Linear, angular.
Attitude: Roll, pitch, yaw.

The conceptual structure of the core ontology focuses on the AMR system. The following classification of concepts describes the structure of the core ontology:

- System context
  - Environment (sensing/actuating)
  - Stakeholders (end user interface)
  - Other systems
- System architecture
  - Structural description
    - Composition
      - Data (observation + ...)
      - Software (modules, services, agents)
      - Hardware
      - Mechanics

- Topology (JAUS; systems, subsystem, nodes, components)
- Messages (JAUS)
- System platform elements (group of vehicles, vehicle, device, transducer)

- Behavioural description
  - Transitions (Events)
  - States
  - Processes (of services or activities; mission, operation, task, action)

System mission

- Goals
- Plans
- Capabilities (including payload)
- Targets (physical objects)

System status

Figure 3.9 shows the main classes of the Core Ontology. This class is the entry point to the core ontology since the concepts shown are connected to (depend on) the central entity which is called “thing”. This means that any entity (or concept) is a thing. Each of these main entities is developed in details in Appendix A.

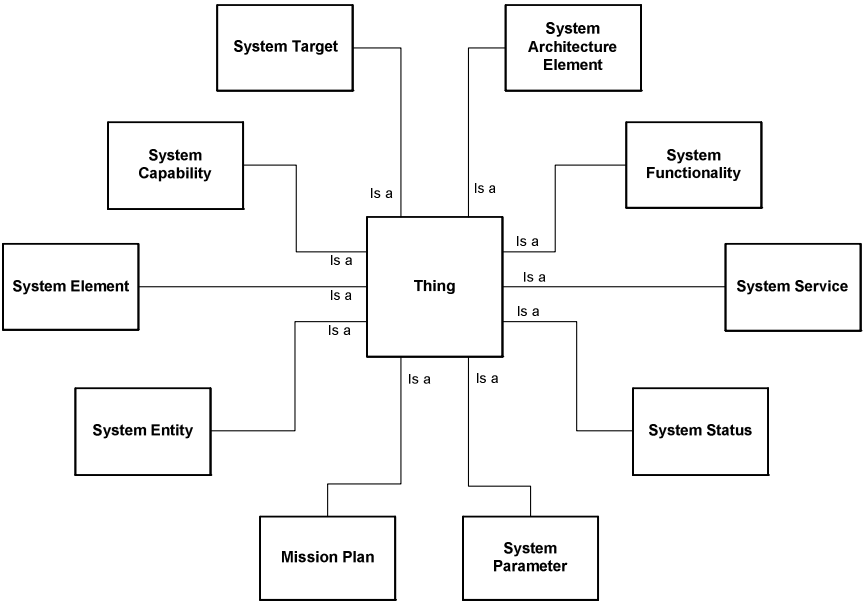


Fig. 3.9 Main classes of the core ontology

Figure 3.10 shows the relations (properties) among the core ontology individuals (focus on capabilities). The most important cross-entity relation is that between 'system capability' and 'system mission'.

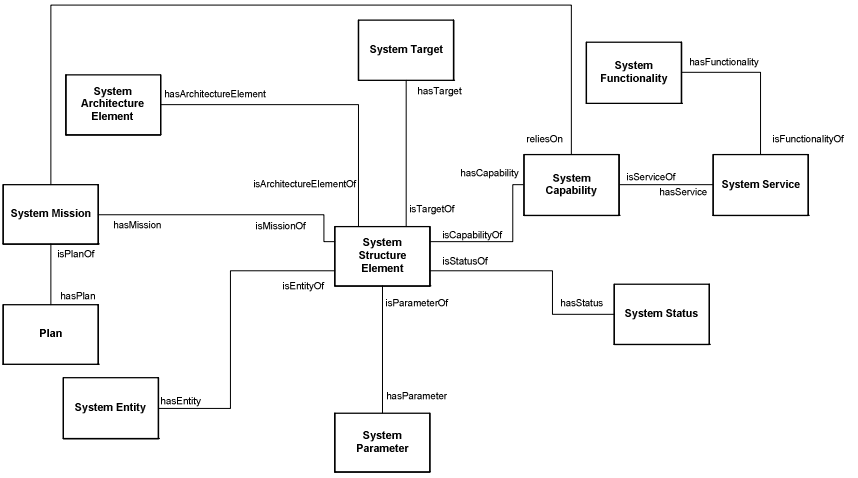


Fig. 3.10 Relations (properties) among the core ontology individuals (focus on capabilities)

Application Ontology

Application concepts are handled at the executive layer and are used to ground the abstract concepts managed by the agents running in the vehicle. Application concepts are specific to the expertise or service provided by each of the intelligent agents. In the case study presented in this book, these agents are the OCU, ASC, and IAUV. These agents make use of the proposed framework and allow the transition from the Deliberative to the Action phase of the OODA loop [25].

The most important concepts identified for service-oriented distributed mission planning are:

Resource: state of an object (physical or abstract) in the environment (vehicle, position, sensor, etc.)

Action: Capability to modify the state of resources (calibrate, classify, explore, etc.)

Plan: A list of time slots containing sequences of instantiated actions

Execution: When an action instantiation is executed successfully

The design of the ontologies encapsulating the knowledge handled by the above agents is described as follows.

The conceptual structure of the application ontology focuses on the AMR system mission. The following classification of concepts describes the structure of the application ontology:

Missions

- Goals
- Plans
- Core ontology: Capabilities
- Activities
  - Messages (commands linked to platform capabilities)
  - Core ontology: Services

Figure 3.11 provides a global and extensible model into which data originating from distinct sources can be mapped and integrated. The knowledge representation in this level is given by a common set of architecture elements and concepts from JAUS.

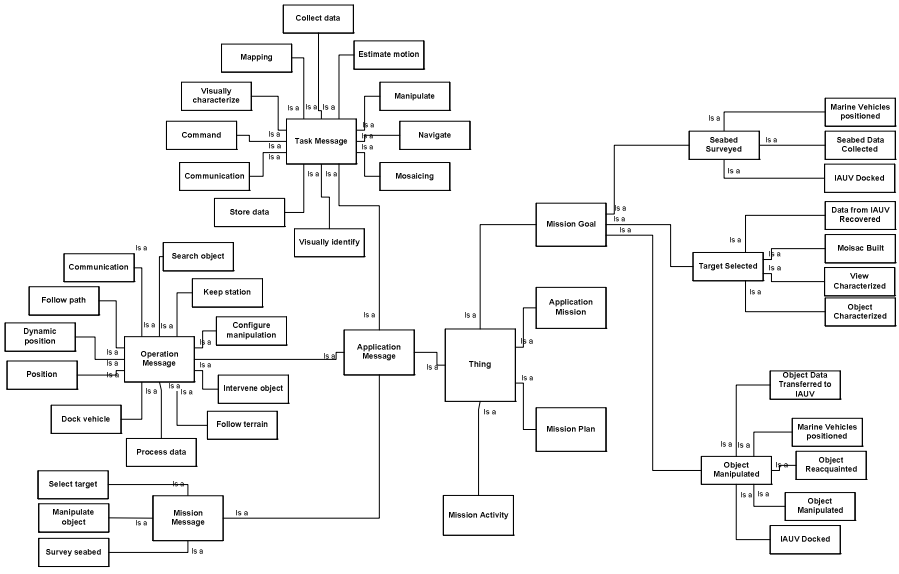


Fig. 3.11 Application ontology

The application ontology (Figure 3.12) provides an underlying formal model for tools that integrate source data, and perform a variety of extended functions. The application concepts are handled at the executive layer and are used to ground the abstract concepts managed by the agents running in the vehicle.

Application concepts are specific to the expertise or service provided by each of the intelligent agents. In the case study presented in this book, these agents are two marine vehicles, and the operator console. These agents make use of the proposed framework and allow the transition from the Deliberative to the Action phase of the OODA loop [23].

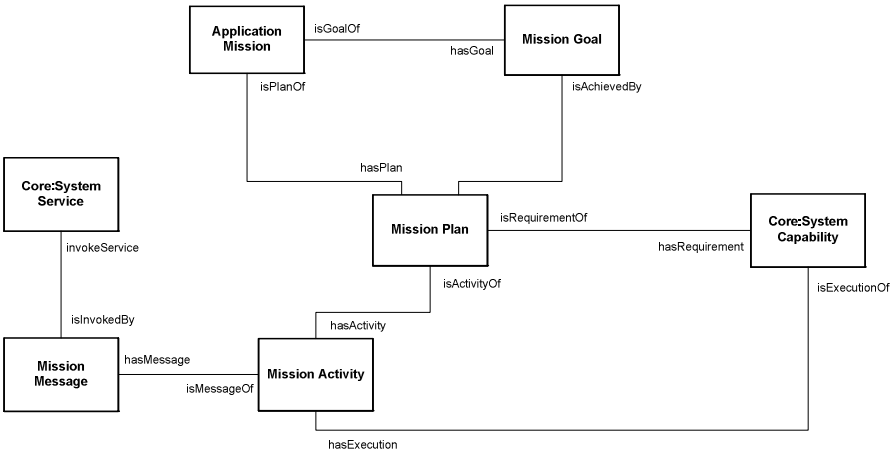


Fig. 3.12 Relations (properties) among the application ontology individuals

3.4 Knowledge Reasoning

The human operator sets the mission to be carried out through the OCU. He/she commands this order to communicate to the maritime vehicles (ASC and IAUV), through the mission communicator to the mission planner, the mission assigned. After setting the mission to be carry out, many questions come up. The first question is that to know whether or not a maritime vehicle is really able to carry out such a mission of part of it. The answer to this question comes from the maritime vehicle that responds based on knowledge about itself as a potential platform suitable (capabilities represented by the pool of services) of successfully performing the tasks required. Then, the following questions are: what capabilities are required from the vehicle platform? Can the vehicle do the job (mission) in a time period? Etc. These questions are made by means of the reasoner that interacts with the mental model in order to know the answer. Then, the answer is passed to the mission planner which begins to make the plan based on the information obtained from reasoning and the social and geospatial models.

Initially, two possible approaches for planning based on knowledge representation (same ontological database for the OCU, ASC, and IAUV) are proposed:

Built-in Plans. Description of the predefined plans in the ontological database, retrieval of the plan, and then checking capabilities supported by the vehicles to execute the plan.

Built-on-Demand Plans. Build the plans based on queries performed against the ontological database by using a forward search algorithm. Then, check consistency against the capabilities available in the system platform.

3.4.1 Built-in Plans

The queries to be performed against the ontological database in order to deal with built-in plans are in the following order.

1. Does the marine vehicle have any predefined plan to tackle the mission operation given? If so, retrieve the plan, and go to the next query. If not, the marine vehicle is not able to carry out the mission operation due to lack of plan, and then notify it to the rest of the system. The query select statement to answer this question is as show in Query 1.

$$\boldsymbol{SELECT}\ ?Mission\ ?Operation\ ?Plan$$
$$\boldsymbol{WHERE}\ \{applicat0ion{:}\ hasOperation(?Mission\ ?Operation)\ \wedge$$
$$application{:}\ hasPlan(?Operation)$$

Query 1. Formal search sentence for predefined plan.

2. Does the marine vehicle have the capabilities to implement the plan retrieved? If so, return successfully, and go to the next query. If not, the marine vehicle is not able to carry the mission operation out due to lack of one or more capabilities need, and then notify it to the rest of the system. The query select statement to answer this question is as show in Query 2.

$$\boldsymbol{SELECT}\ ?Platform\ ?Capability$$
$$\boldsymbol{WHERE}\ \{rdf{:}\ type(\ ?Platform, core{:}\ Platform)\ \wedge$$
$$core{:}\ hasCapability(?Platform\ ?Capability)$$

Query 2. Formal search sentence for capabilities in the platform (marine vehicle).

3. What are the pre-conditions and post-conditions of every plan activity? Retrieve pre-conditions and post-conditions according to the activities specified in the plan. The query select statement to answer this question is as show in Query 3.

$$\boldsymbol{SELECT}\ ?Activity\ ?Plan$$
$$\boldsymbol{WHERE}\ \{application{:}\ hasPrecondition(?Activity\ ?Plan)\ \wedge$$
$$application{:}\ hasPostcondition(?Activity\ ?Plan)$$

Query 3. Formal search sentence for pre and post conditions of activities.

The reasoning algorithm for the built-in planning is shown in Algorithm 1 where m is a mission, s is a state reached in a plan, π is a plan, c is a capability, o is an operation, a_i is the ith activity, A is a set of activities, *prec* is a pre-condition, and *postc* is a post-condition.

Algorithm 1. Reasoning logic for the built-on planning.

$s \leftarrow s_0$
$\pi \leftarrow \emptyset$
$c \leftarrow \emptyset$
$\pi \leftarrow query1(m, o, \pi)$
while s $\neq g$ do {
  if $\pi \neq \emptyset$ then
    for each $c.a_i \in \pi$ do {
      $available \leftarrow query2(c.a_i)$
      if $available$ then {
        $c.a.prec \leftarrow query3(c.a_i, \pi).prec$
        $c.a.postc \leftarrow querey3(c.a_i, \pi).postc$
        $s = c.a.postc$
      }
    }
  $return\ failure$
}

3.4.2 Built-on-Demand Plans

The queries to be performed against the ontological database in order to deal with built-on-demand plans are in the following order.

1. Does the marine vehicle have any plan to tackle the mission operation given? To answer this question a search algorithm performs queries on the ontological database in search of activities that satisfy the intermediate goals placed between the initial goal and the end goal (mission goal). The first activity chosen is one that has the initial goal as a pre-condition, the second activity is one that has the post-condition of the first one as a pre-condition, and so on. Thus, sub-goals are chained by listing activities in a certain order. If it is possible to go from the initial goal to the end goal by means of selecting activities, then a plan can be defined; go to the next query. If not, the marine vehicle is not able to carry the mission operation out due to lack of a plan, and then notifies the rest of the system. The query select statement to answer this question is as show in Query 4.

$$\boldsymbol{SELECT}\ ?Prestate\ ?Activity$$
$$\boldsymbol{WHERE}\ \{application{:}\ hasActivity(?Activity\ ?Prestate)\}$$

Query 4. Formal search sentence to build the plan

2. Does the marine vehicle have the capabilities to implement the plan retrieved? If so, return successfully, and go to the next query. If not, the marine vehicle is not able to carry the mission operation out due to lack

of one or more capabilities need, and then notify it to the rest of the system. The query select statement to answer this question is as show in Query 5.

$$\textbf{\textit{SELECT}}\ ?Platform\ ?Capability$$
$$\textbf{\textit{WHERE}}\ \{rdf{:}type(\ ?Platform, core{:}Platform)\ \wedge$$
$$core{:}hasCapability(?Platform\ ?Capability)$$

Query 5. Formal search sentence for capabilities in the platform (marine vehicle).

The reasoning algorithm for the built-on-demand planning is shown in Algorithm 2.

Algorithm 2. Reasoning logic for the built-on-demand planning.

$s \leftarrow s_0$
$\pi \leftarrow \emptyset$
$A \leftarrow \emptyset$
$c \leftarrow \emptyset$
while $s \neq g$ do {
  $A \leftarrow query4(s, a)$
  if $A \neq \emptyset$ then $return\ failure$
  $nondeterministically\ choose\ a \in A$
  $s \leftarrow a.postc$
  $\pi \leftarrow \pi.a$
}
for each $a_i \in \pi$ do {
  $available \leftarrow query5(c.a_i)$
  if $not\ available$ then $return\ failure$
}

After answering the above questions, and in either planning approach, the mission reasoner gets back to mission planner in order to generate the plan.

3.5 Goal-Driven Capability-Based Planning

The planning paradigm is time-constrained with activity scheduling according to resource availability. The planning nature comes from classical planning with classical representation [31]. In addition, the planning control strategy is based on Hierarchical Task Network (HTN).

The initial proposal chosen to approach the internal agent planning is very simple. It is inspired by classical approach such as the state-space planning with forward search. The main difference between the classical planning approach considered and the one proposed is that the search mechanism is replaced by a more complex paradigm of search based on reasoning.

Figure 3.13 shows a comparison between the above planning approaches. A block diagram corresponding to the classical planning is shown on the left and a block diagram corresponding to the planning proposed is shown on the right of the figure.

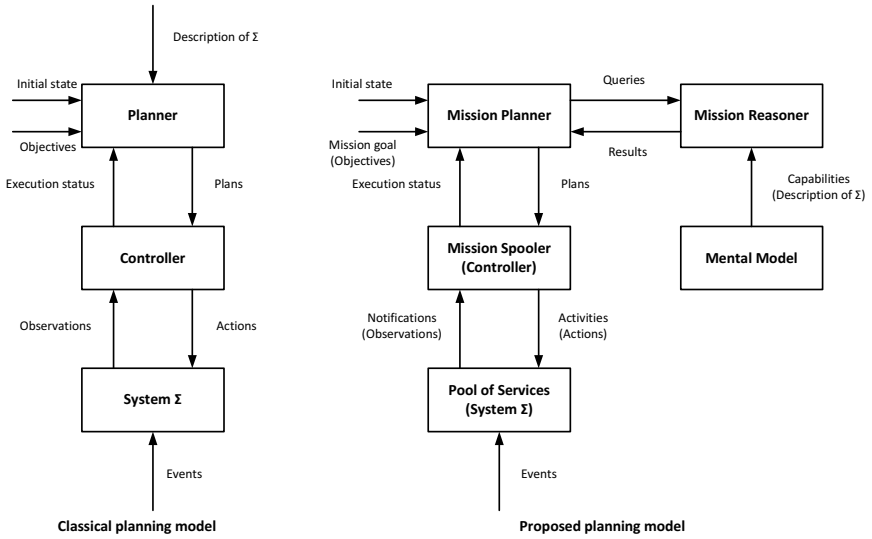


Fig. 3.13 Conceptual planning model comparison

The main difference between the two planning approaches is that for the description of Σ (set of plans). In a traditional planning model (on the left of Figure 3.13) it is set by the user of the system (AMR system operator). In the proposed planning model such description is embedded in the system (AMR system) as knowledge in the ontological database. The system controller is instructed by the planner to carry out the task-based plan through activities (actions).

The description of the AMR system is given by the ontological database. The reasoner queries this ontology in search of solutions for the planning problem. The initial state of the system is given by the initial states of the marine vehicles, i.e. ASC and IAUV. The objectives, in a more general way are represented by goals, where the main goal matches the mission goal that can be divided into sub-goals.

Once the plan is initially pre-defined with the information obtained from the ontological database, the mission planner checks the plan consistency in terms of capabilities available in the system platform.

3.6 System Development Process

The system development process has the following typical phases: requirements analysis (requirements definition, system architecture specification), system design (architecture design), system realization (architecture realization; Chapter 5), and system verification/validation (architecture evaluation; Chapter 6).

Figure 3.14 shows the design methodology for the vehicle intelligent control architecture. It involves the first three stages of the development process (user and system requirements definition, system architecture specification, and system architecture design). The remaining two stages of the development process are the system implementation (based on ROS), and the system verification and validation (evaluation scenarios).

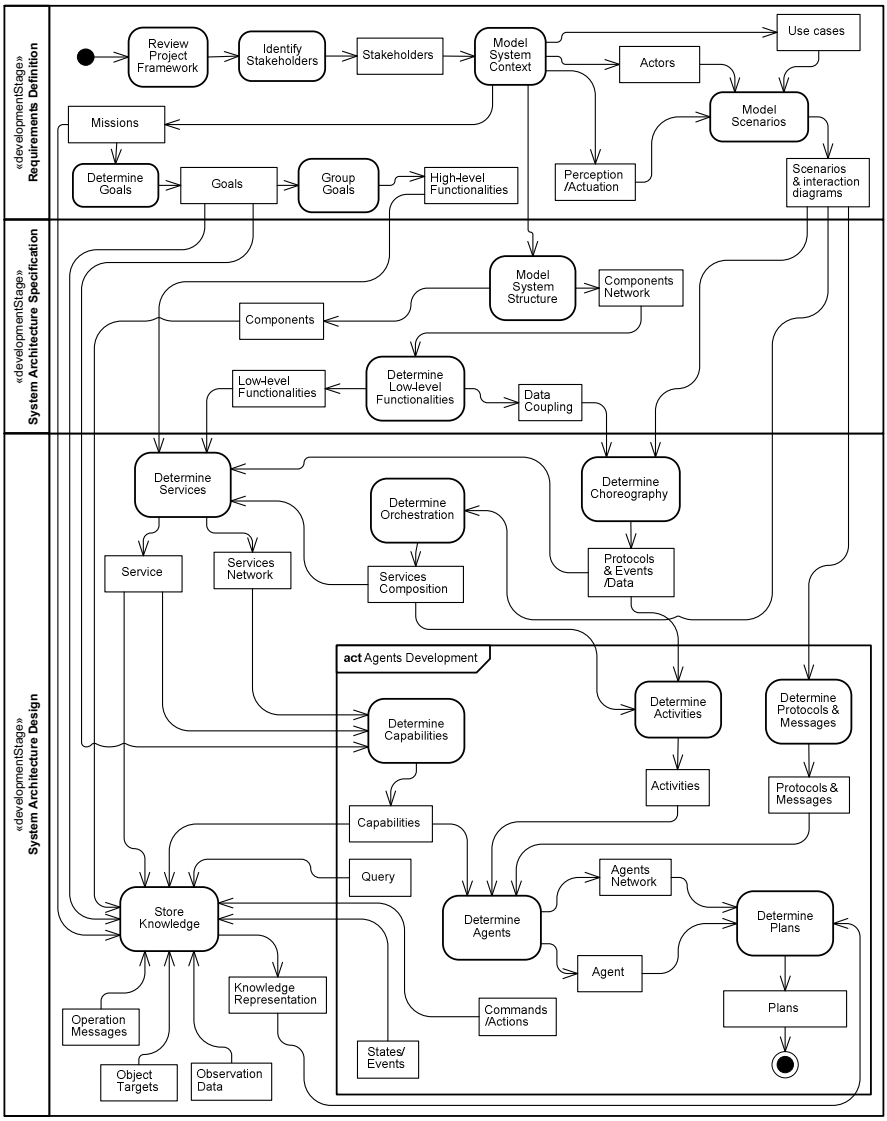


Fig. 3.14 First phases of the system development process

As shown in Figure 3.14, the development process is divided into two parts. The first part involves a service-oriented approach, and the ontology design. The second part (Agent Design) involves an agent-based approach which is built on the previous approach in order to finally get a service-oriented agent-based architecture for the AMR system.

User and System Requirements Definition (top of Figure 3.14). The definition of the requirements consists of statements about the functionality expected from the system and the constraints under which it must operate. The outcomes obtained from this phase are:

- Intermediate results (white rectangle boxes)
    - Identification of Stakeholders
    - System context diagram
    - Identification of actors
    - External system interfaces (environment, stakeholder, other systems)
    - Use cases
- End results; incoming information required by the next phases (grey rectangle boxes)
    - Scenarios (interaction diagrams from use cases)
    - Perception/Actuation
    - Missions, goals and high-level functionalities

System Architecture Specification (middle of Figure 3.14). The specification of the SOA consists of defining in details of all the available services in the system. It is based on the information provided by the user and system requirement definition. The outcomes obtained from this stage are:

- Intermediate results (white rectangle boxes)
    - System structure diagram
    - Components network (components and connectors)
    - Low-level protocols (from system components)
- End results; incoming information required by the next phases (grey rectangle boxes)
    - System components
    - Low-level functionalities (from system components)
    - Data coupling among components

System Architecture Design (bottom of Figure 3.14). The design the SOA consists of describing all the above services by focussing on their orchestration and choreography. The former deals with the composition of service process (multiple encapsulations at runtime). The latter deals with the interaction among services processes when they are executed in parallel (how they exchange information, i.e. messages, protocols and models of communication).

- Intermediate results (white rectangle boxes)
    - Orchestration and choreography of services (Interactions among services; protocols)
- End results; incoming information required by the next phases (grey rectangle boxes)
    - Services (interfaces; data & events). Basic services (from low-level functionalities) & composite services (from low-level functionalities)
    - Communication patterns for communication among services

Knowledge representation (information collected from the different development phases).