Stable Analysis Patterns for Robot Mobility

Davide Brugali

Universitá degli Studi di Bergamo, Italy `brugali@unibg.it`

1 Introduction

During the last few years, many ideas for software engineering (modularity, information hiding, Component-based development, and architectural styles) have progressively been introduced in the construction of robotic software systems, to simplify their development, and to improve their reusability. All of these techniques offer partial (sometimes overlapping) views and solutions to the problem of developing reusable software.

While a universal reuse solution remains elusive, great improvements can be made by focusing on well-defined areas of knowledge (domains). The term domain is used to denote or group a set of systems (e.g. mobile robots, humanoid robots) or functional areas (motion planning, deliberative control), within systems, that exhibit similar functionality.

Domain Engineering is a set of activities aiming at developing reusable artifacts within a domain. The fundamental tenet of Domain Engineering is that substantial reuse of knowledge, experience, and software artifacts can be achieved by performing some sort of commonality/variability analysis to identify those aspects that are common to all applications within a domain, and those aspects that distinguish one application from another [CHW98].

Recently, the focus of domain engineering has moved from commonality/variability analysis, toward the new concept of stability analysis. Software stability can be defined as a software system's resilience to changes in the original requirements specification. Stability analysis enhances reuse as it focuses on those aspects of a domain that will remain stable over time. Such an approach ensures a stable core design, and thus stable software artifacts [FA01].

In order to support stability analysis, the concept of an Enduring Business Theme (EBT) was first introduced in [CG00], and further investigated in [Fay02] and other papers by the same authors. An EBT is an aspect of a domain that is unlikely to change, since it is part of the essence of the domain. For example, the robot property of having a physical body is a an essential aspect of every robotics application. An EBT is modeled as software entity that represents the stable property at a high level of abstraction. At design level, EBTs are refined by more concrete software entities called Business Objects (BO), which offer application-specific functionality. BO have stable interfaces and relationships between them. At implementation level, BOs are internally implemented on top of more transient software entities called Industrial Objects (IO). The identification of EBTs, BOs, and IOs, requires a deep knowledge and understanding of the domain.

In a previous paper [BR05], we have formulated the problem of developing stable software systems in the mobile robotics domain, and analyzed the issues that make the problem hard. In [BS06] we have identified three Enduring Business Themes related to Embodiment, Situatedness, and Intelligence, and we have modeled them as stable analysis patterns. In this chapter, we present three Analysis Patterns that document these three EBTs and a framework of BOs that refine at design level the Embodiment EBT.

The three Analysis Patterns are documented using the the template defined in [CHF05], which is structured in four sections. The *Context* describes the possible scenarios for situations in which a pattern may recur. The *Problem* presents the problem a pattern concentrates on. The *Solution* illustrates a stable object model in terms of participants and relationships. There are two main participants in a solution model: classes and patterns. Classes are the same as those which are used in traditional Object-Oriented class diagrams. Patterns are themselves models that contain classes, and in some cases other patterns. In the model, BOs names start with "Any" indicating that they are standalone stable patterns, and satisfy any application. The fourth section, *Consequences*, analyzes the trade-off and results of using a pattern.

The remainder of this chapter is organized as follows: Section 1.1 introduces three stable aspects related to robot mobility. Sections 2, 3, and 4, present the Intelligence EBT, the Situatedness EBT, and the Embodiment EBT respectively. Section 5 documents the design of the BOs that refine the Embodiment EBT. Finally, Section 6 draws relevant conclusions, and highlights future steps for this research.

1.1 Robot Mobility

A milestone paper by Rodney Brooks [Bro91], identifies a set of properties for every mobile robotics system, among which three are of interest in our discussion:

Intelligence: Robot intelligence refers to the ability to express adequate and useful behaviors while interacting with a dynamic environment. Intelligence is perceived as "what humans do, pretty much all the time" [Bro91]; and mobility is considered at the basis of every ordinary human behavior.

Robot mobility is related to Intelligence as explained in terms of robot behaviors and tasks. These concepts help to answer two questions: How does a robot move? Why does a robot move?

Situatedness: Robot situatedness refers to robots existing in a complex, dynamic and unstructured environment, which strongly affects the robots behavior. For example, the environment could be a museum full of people where a mobile robot guides tourists and illustrates masterworks, or a game field where two robot teams play soccer, or a manufacturing workcell where an industrial manipulator handles workpieces. Situatedness implies that a robot is aware of its own posture in one place at a given time. It does not deal with abstract descriptions, but with the here-and-now of the environment that directly influences robot behavior.

Robot mobility is related to Situatedness, as it is a form of robot-environment interaction. It refers to the ability of a robot to change its own position with regard to the environment, and thus of being in different places at different times. Situatedness helps to answer two questions: With respect to what does a robot move? When and where does it move?

Embodiment: Robot embodiment refers to the consciousness of having a body (a mechanical structure with sensors and actuators) that allows a robot to experience the world directly. A robot receives stimuli from the external world, and executes actions that cause changes in world state. Simulated robots may be 'situated' in a virtual environment, but they are certainly not embodied.

Robot mobility is related to Embodiment, as it applies to both a robot and its constituent components, which move with respect to each other. For example, a humanoid robot is a complex structure with many limbs that change their relative position while it is walking. This also applies to parts or devices of a robot that move with respect to the environment: For example, when a robot stands but changes its head orientation to track a moving object (e.g. a human being).Embodiment helps to answer two questions: Which part of a robot does move? How much does it move?

Simon's "ant on the beach" [Sim69], is a classical example which demonstrates the intrinsic relation of mobility with Intelligence, Situatedness, and Embodiment. An ant's behavior control mechanism (its intelligence), is very simple: obstacle right, turn left; obstacle left, turn right. On a beach with rocks and pebbles (situatedness), an ant's trajectory will be a zigzag line (mobility). But if the size of an ant (embodiment) were to be increased by a factor of 1000, then its trajectory would be much straighter.

2 Intelligence Analysis Pattern

Robotics is an experimental science that can be analyzed from two perspectives. From one perspective it is a discipline that has its roots in mechanics, electronics, computer science and cognitive science. In that regard, robotics plays the role of an integrator of the most advanced results in order to build complex systems. From another perspective, robotics is a research field which pursues ambitious goals, such as the study of intelligent behavior in artificial systems. Many achievements have found application in industrial settings and everyday life.

Context

The concept of intelligence (the ability of expressing useful behavior) is quite elusive. It is usually associated with other concepts, such as autonomy, i.e. a robot's ability to control its own activities, and to carry out tasks without the intervention of a human operator [Ark98] (e.g. navigating towards a target position without colliding with obstacles); deliberativeness, i.e. the ability to plan and revise future actions in order to achieve a given goal while taking into account the changeable conditions of an external environment [KDR04] (e.g. planning the shortest path between two locations in an indoor environment); and adaptability, i.e. the ability to change behavior in response to external stimuli, according to past interactions with the real world (e.g. recognizing already visited places).

Problem

The key ingredients in describing robot autonomy (deliberativeness, and adaptability, and thus the ability to move), are the actions that a robot executes, the behaviors that it exhibits, and the control schema that it implements.

The first goal of this pattern is to define these three concepts and to identify their relationships.

Robot intelligence is enacted by its control architecture. In [Mat02] Maja Mataric summarized the concept of robot control as "the process of taking information about the environment, through the robot's sensors, processing it as necessary in order to make decisions about how to act, and then executing those actions in the environment". This definition implicitly refers to the concepts of situatedness and embodiment.

The second goal of this pattern is to insulate software abstractions that refer to robot control from those that refer to robot-environment interaction and robot structure. This will allow robot control architectures to be developed, which are not tied to any specific robot platform, so that different control architectures can be experimented with on the same robot platform.

Solution and participants

A solution consists of the definition of the Intelligence EBT, and a set of related analysis patterns (BOs). A stable object model is depicted in Figure 1. Three BOs represent stable abstract concepts that characterize robot intelligence.

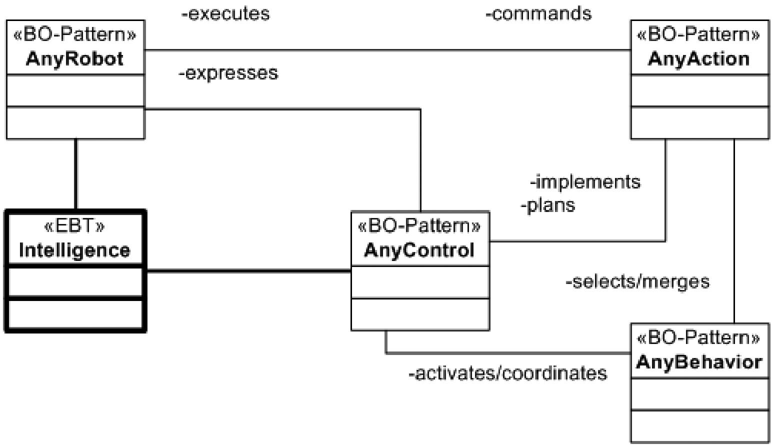

Fig. 1. The Intelligence Analysis Pattern

BO: AnyAction - This represents a generic action that a robot can execute. Robot actions can be defined as those activities which produce modifications in a robot's hardware configuration and cause robot-environment interaction. For example, actions are commands issued to a robot's actuators in order to change the relative position of rigid components, and therefore their position with regard to the external environment. Actions can also include the activation/deactivation of a robot's sensors by which physical interactions with an environment can be determined (e.g. the emission of a sonar pulse that detects the proximity of an obstacle).

The AnyAction pattern is intended to define an abstract representation of a robot's actions, independent of a specific robot structure and its equipment. Key features of this representation are for example timing requirements, expected results, a probability of success, possible types of fault, and recovery procedures.

BO: AnyBehavior - This represents the generic behavior that a robot can exhibit. A robot behavior is a schema of actions that interpret the current status of the robot, environment, and external stimuli, in order to select the next command to issue to a robot.

The AnyBehavior pattern is intended to define an abstract representation of a robot's behaviors which is independent of the specific sensor and actuator system of the robot. Behaviors are constructed by composing available robot actions, according to possible robot-environment interactions.

BO: AnyControl - This represents generic control schema that a robot can perform. Control of robots is one of the traditional testbeds for several areas of artificial intelligence, such as problem solving, planning, learning and coordination, etc. Four main classes of robot control methodology have been identified for mobile robots: reactive, deliberative, hybrid, and behavior-based [Mat02]. All of these can be described in terms of actions and behaviors.

Reactive control is a technique for tightly coupling sensor data and command actions. It builds on the concept of closed-loop feedback, derived from control theory. For example, the ant's control law [Sim69] is described by two simple perception/action patterns: obstacle at right, turn left; obstacle at left, turn right.

Deliberative control refers to the use of symbolic reasoning in classical Artificial Intelligence. A robot constructs and evaluates alternative sequences of actions (plans), and executes the one that best satisfies its constraints and goals.

Hybrid control merges the previous approaches by coordinating immediate responses to external stimuli, whilst also executing goal-directed planning.

Behavior-based control deals with the activation and coordination of robot behaviors that simultaneously govern robot motion in an environment. For example, continuous robot motion can be determined by a simultaneous fusion of command actions that guide a robot towards a goal (e.g. a moving ball) and away from the obstacles (e.g. other robots).

Consequences

The Intelligence Pattern has the following pros and cons:

Mobility-oriented: it focuses only on those aspects of robot intelligence that are more related to mobility, i.e. on how to control a robot in order to exhibit a given behavior.

Completeness: it analyzes the conceptual building blocks of four classical robot control paradigms, i.e. reactive, deliberative, hybrid, and behavior-based.

Flexibility: it provides a clear separation of aspects related to actions, behaviors, and control, which allows the development of flexible robot architectures that encompass the entire spectrum of robot control architectures.

3 Situatedness Analysis Pattern

Robot situatedness means that a robot is located in an environment in which it operates, both from spatial and temporal points of view. The aim of a Situatedness Pattern, is to capture knowledge related to the here and now aspects of the interaction between a robot and its environment.

Context

Robotic devices have been conceived and developed to operate in a variety of environments. Service and humanoid robots operate in indoor environments designed for and occupied by humans. Industrial robots perform repetitive tasks in structured manufacturing plants. Space robots explore the surface of planets and satellites, or inspect and repair space shuttles. These are just a few examples.

Robots interact with their operating environment according to a variety of spatial-temporal patterns, which depend on the type of task the robot has to perform. Both time and space can be represented as continuous or discrete dimensions. For example, the trajectory traveled by a mobile robot is a continuous spatial-temporal interval; past, current, and future robot locations are discrete spatial-temporal positions.

Problem

Robot systems are usually developed both from a hardware and software point of view for a given target environment. Nevertheless, many design solutions conceived for highly different environments share relevant commonalities, at least from a conceptual point of view. For example, the concept of obstacle avoidance is the same, whether a robot has to avoid rocks on the surface of Mars, or wastebaskets in an office. Therefore patterns are used to concentrate on the problem of capturing the basic elements, and relationships, among these elements of robot situatedness; in order to develop a conceptual foundation for the description of every robot environment.

Solution and Participants

A solution consists in the definition of a Situatedness EBT, and of a set of related analysis patterns (BOs). A stable object model related to a Situatedness EBT is depicted in Figure 2; it shows participants (BOs), and the relationships between them. The BOs represent stable abstract concepts that characterize robot situatedness. In order to analyze and design concrete robot systems, every BO has to be further detailed in order to capture the requirements of dynamic robot-environment interactions (e.g. real-time constraints, historic data log, etc.); and can be documented as a pattern by themselves.

EBT: Situatedness - This represents the concept of the situatedness of any robot operating in an environment. This class consists of behaviors and attributes that allow the customization of a system with regard to global properties that cut across individual BOs, such as the role and characteristics of an external observer.

BO: AnyRobot - This is the actor that expresses robot behaviors while it interacts with an environment. Robot-environment interaction is governed by three major components: a) a robot itself, its sensors and actuators, b) an environment's perceptual properties, and c) a task, usually a control program being executed by a robot. A robot's behavior emerges through the interaction of these three aspects. We intentionally concentrate our attention only on the role of a robot as an active player in environmental interactions. Other

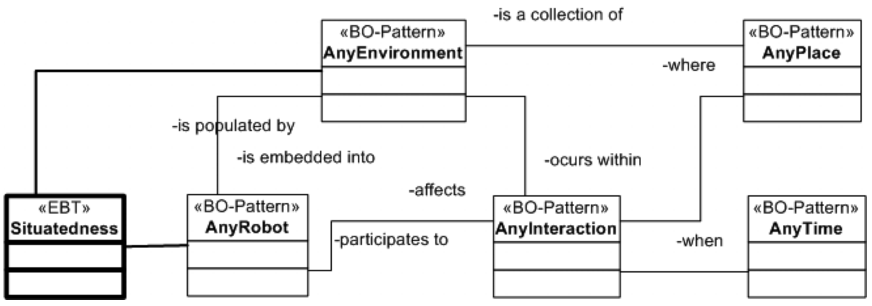

Fig. 2. The Situatedness Analysis Pattern

players, such as a human operator, or other dynamic systems populating an environment, have to be individually described by other BOs, and patterns that are beyond the scope of this chapter. The relationship between a model's participants, state that AnyRobot is embedded in AnyEnvironment, and participates in several AnyInteractions.

BO: AnyInteraction - This represents the type of interaction that occurs between a robot and environment. Basically, we can classify interactions into two main categories: physical and behavioral interactions. Moving within a room, grasping an object, and measuring physical quantities such as distances, forces, and energy are examples of physical interactions. Physical interactions are non-deterministic, at least from a robot's point of view, because sensors and actuators are imperfect, therefore measurements and actions are affected by uncertainty. Complex combinations of physical interactions can be abstracted into behavioral interactions. For example, supervision is a kind of human-robot interaction that requires a robot to interact with a human operator via a variety of media, such as speech, vision, force, etc. Collaboration and competition are typical robot-to-robot interactions, while learning is a high level robot-environment interaction. The relationships between a model's participants, state that AnyInteractions affect the behavior of AnyRobot, can occur in AnyEnvironment, and take place in AnyPlace and at AnyTime.

BO: AnyPlace - This represents where an interaction takes place. It can be meant as the portion of an environment affected by an interaction. For example, a humanoid robot walks on a floor, or a manipulator robot operates in a workcell. The computational representation of places, depends on the tasks that robots have to perform in the environment, and on the kind of interactions that have to be modeled. For example, a robot manipulator needs a detailed 3D geometric representation of its operational environment in order to accurately grasp an object, while an autonomous mobile robot is able to navigate in an indoor environment by localizing with relation to known landmarks. The relationships between a model's participants, state that AnyEnvironment is a collection of AnyPlaces which always belong to AnyEnvironment. AnyPlace is where AnyInteraction can occur, which is always spatially situated in AnyPlace.

BO: AnyTime - This represents the time at which an interaction takes place. The timing characteristics of a robot-environment interaction can be described basically in terms of discrete events, or continuous intervals that determine its occurrence (e.g. a clock tick fires a sonar reading, operator input stops a movement, a grasping tool holds a workpiece for 10 seconds, etc.). The relationship between a model's participants, state that AnyTime is when AnyInteraction occurs.

BO: AnyEnvironment - This represents the totality of the surrounding physical conditions that affect, enable and limit a robot, while it is performing a task. The environment is a continuum of physical configurations, but from a computational point of view it can be represented as a discrete spatial-temporal milieu, that is, made up of robots and any other dynamic or static system such as people, equipment, buildings, all of which have mutual interactions governed by the laws of Physic's. An environment is partially observable due to the limited capabilities of a robot's sensors (vision systems do not work in a dark environment, laser range finders do not detect transparent surfaces, etc.). An environment is unstructured, as it has usually not been specifically designed to host a robot in order to simplify task execution. An environment is unpredictable since it is populated by systems whose dynamic behavior is not under a robot's control. The relationships between a model's participants, state that AnyEnvironment is populated by AnyRobot, and is a collection of AnyPlace and hosts AnyInteraction.

Consequences

The Situatedness Pattern has the following pros and cons:

Abstraction: it captures the fundamental elements and relationships that are needed to describe the situatedness of every robot in every environment, but has to be refined to capture the qualitative and quantitative factors that characterize any specific situated robot.

Extensibility: a pattern is modular and allows the seamless introduction of new concepts, such as those of human operator and generic dynamic systems.

Common language: a pattern facilitates the easy description of a variety of systems, developed by different researchers, using a common language.

4 Embodiment Analysis Pattern

The concept of embodiment was thoroughly investigated during the second half of the last century by many researchers in Cognitive Science, Artificial Intelligence, and Robotics (see [CZ00] for a survey).

In the context of our discussion, a robot's physical body, i.e. its electro-mechanical structure, is a medium employed to experience a sense of time and space, i.e. by being situated in a physical environment.

Context

Physical interactions with the surrounding world occur via sensors and actuators. Sensors allow a robot to perceive environmental changes and to react by changing its own behavior (e.g. the detection of a very close obstacle causing a robot to switch from an operating to an emergency mode). By means of its sensors, a robot experiences the effect of its actions on an environment via the effects that such actions produce. Actuators are the tools with which a robot changes its environment. The spatial-temporal interaction of sensors and actuators with an environment is a consequence of their physical situatedness, which is dependent on a robot's structure.

Problem

Robots substantially differ from one other in their mechanical structure. Nevertheless robots are heterogeneous compositions of a limited number of basic building blocks (sensors, motors and actuators, control and processing units, and communication interfaces). Robot control applications strongly depend on the type of robot used to carry out a task, i.e. a robot's mechanical structure greatly influences the dynamics of its body structure and the requirements of software applications that control it. For example, a video camera mounted on a mobile robot can be used for site surveillance, while the same video camera mounted on a room ceiling can help a robot to self-localize.

This pattern concentrates on the problem of capturing the basic elements and relationships between the elements of robot embodiment, in order to develop a conceptual framework for a description of all robot mechanisms. Therefore, the goal of this pattern is to define mechanical abstractions, which enable the development of stable software that can be used to control and expose robot hardware.

Solution and participants

A solution consists of the definition an Embodiment EBT, and a set of related analysis patterns (BOs). A stable object model is depicted in Figure 3. Three BOs represent stable abstract concepts that characterize robot embodiment.

BO: *AnyBody* - This represents the electro-mechanical body of any robotic mechanism, i.e. the ability to perceive its own state in terms of relative position, orientation, and movement of a robot's body and its parts, level of battery charge, fault conditions of its devices (proprioception), and the ability to change its internal state and interaction with the external world

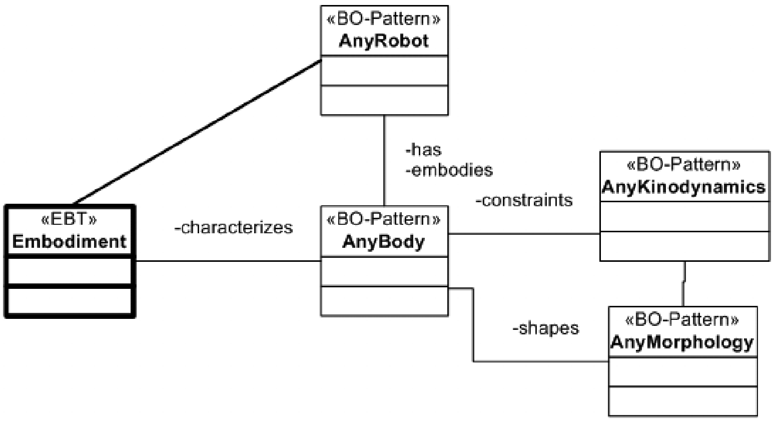

Fig. 3. The Embodiment Analysis Pattern

(actuation). The relationships between the model's participants, state that *AnyBody* embodies AnyRobot.

BO: AnyMorphology - This describes the shape of a robot's body, and its components and their structural relationships. Typically, robot mechanical design is based on serial link chains with actuated rotary or prismatic joints (industrial manipulators). More advanced bio-inspired robotic structures include both active and passive joints, compliant structures, and rigid components connected by a network of tensile elements. Reconfigurable or evolvable robots have the ability to adapt both body and control mechanisms to their environment. Several types of robot structures aim to replicate the morphology of animals (ant, dog, snake), and human beings (humanoid robots). The majority of robots resemble human vehicles: cars, tanks, blimps, aircraft. The relationship between the model's participants, state that AnyMorphology shapes *AnyBody* .

BO: AnyKinoDynamics - This describes the kinematics (position and velocity) and dynamic (acceleration, force) constraints that limit the relative movement of a robot's mechanical components and body in an environment. These constraints are determined by a robot's morphology (e.g. links and joints), by the mechanical power of actuators (e.g. torque force), and by the laws of Physics (e.g. gravity). For example, holonomic constraints reduce the number of degrees of freedom for a set of linked rigid bodies from a number of original coordinates. A holonomic robot is capable of rotating (turning) while moving forward or backward and left or right. In other words, it has a 3 degree-of-freedom base, which is the maximum possible degrees-of-freedom for a mobile robot. The relationships between the model's participants, state that AnyMorphology shapes *AnyBody* .

Consequences

The Embodiment Pattern has the following pros and cons.

Specificity: it focuses only on a specific but fundamental aspect of embodiment, i.e. the physical structure of a robot body. It does not address issues related to the role of the body in cognition, or the structure of an animal brain as a connection between mind and body.

Modularity: it explicitly separates the structural and dynamic features of a robot's body, although they are interdependent. This choice allows the expression of kinodynamics properties of the same robot components in different forms (e.g. geometric or differential equations, logical constraints, etc.) depending on the control application that uses them.

Abstraction: Modularity also facilitates the representation of properties that do not refer to any specific part or component of a robot, but to its body as a whole. For example, despite vastly different morphologies, it is possible to abstract based on functional characteristics the motion of a robot body (e.g. a translation through space).

5 From Analysis to Design: BOs for Embodiment

Robot control applications strongly depend on the type of robot used to carry out a task, i.e. the robot mechanical structure greatly influences the requirements of the software applications that control it. Typically, the mechanical model information are scattered among different functional modules: the locomotor uses the kinematics models of wheels or legs; the obstacle avoider and the path planner need information about the robot shape; the visual odometer stores the position of the stereo vision system with respect to the robot reference frame. As a consequence, most implementations of robot functionality are tied to specific robot hardware, which might substantially differ from one other in their mechanical structure.

In order to make control applications reusable across different robot platforms, there is the need to make software dependencies to the mechanical structure explicit. This means to define a common mechanism model that is used by every functional module to refer to the specific characteristics of the robot hardware. A few research projects have dealt with this issue, as documented in [OSCAR, OROCOS, Nesnas06].

Such a model should enable the description of the structural and behavioral properties of each mechanism part or subsystem, the relationships and constraints among the parts, and the topology of the system. From a software development perspective, the mechanism model should be described in terms of software abstractions that have the following software quality factors:

- *Expressiveness.* The model enables the description of different mechanism types, such as open loop and closed loop chains.

- *Stability.* The model representation is resilient to changes in the application requirements specification.
- *Scalability.* The model allows the seamless composition of simple mechanisms into more complex mechanisms.
- *Efficiency.* The model supports the implementation of efficient query and update operations.

In this section we illustrate the ANYMORPHOLOGY Design Pattern which describes a set of software abstractions used to model robotic mechanisms. The model has been used to develop control applications of both industrial manipulators and mobile rovers.

5.1 Related Works

Many approaches to mechanisms modeling have been documented in the literature. They adopt different representation paradigms which are tailored to specific application domains, such as multibody systems simulation [Otter96], and 3D mechanical design [SPK00]. A few attempts at defining a unified representation for robot mechanisms have been documented in the literature, such as DARTS/DShell [DDS, Jain91], OSCAR [OSCAR, PTKT02], and RoboML [RML]. The OROCOS [BSK03] and CLARAty [NWBS03, CLR] projects have defined software abstractions to represent any kind of robotic mechanism.

The Open Robot Control Software (OROCOS) project represents an attempt to develop a general-purpose, open-source, modular framework for robot and machine control, whose ongoing development has started in year 2002. Currently, the project has released three software toolkits, which include the "Kinematics and Dynamics Library", and a set of requirement documents, which describe concepts related to the semantic decoupling between physical properties, mathematical representations, symbolic attributes, and interconnection information. In particular, the concept of interconnection decoupling has been described in the "Object-Port-Connector" (OPC) Software Pattern [OROCOS].

The Coupled Layered Architecture for Robotic Autonomy (CLARAty), which has started in 2000, is a framework for generic and reusable robotic components that can be adapted to a number of heterogeneous robot platforms at NASA. It defines and implements the Mechanism Model framework [DNNK06] for modeling mechanisms for robotic control.

5.2 The AnyMorphology Design Pattern

The ANYMORPHOLOGY Design Pattern pursues software stabilities by defining software abstractions that directly map to concepts found in problem domain and by modeling them in the same way as corresponding physical entities are described. Five software abstractions characterize the Pattern: *AnyElement* , *AnyPort* , *AnyPair* , *AnyMechanism* , and *AnyBody* . Their

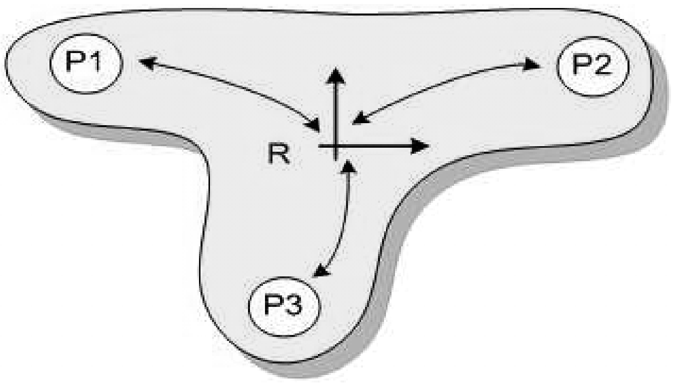

Fig. 4. An example of mechanical element with three ports

names starts with "Any" in order to emphasize their independence of any specific application.

AnyElement represents elemental mechanism parts, such as manipulator links, rover chassis, linear springs, etc. It has one reference frame relative to which all the kinematics properties of an element are expressed. It holds a symbolic attribute describing information related to its physical position on the element, e.g. it is coincident with the centre of mass or with one of its ports. *AnyElement* is intended to store information related to several types of physical properties, such as the mass distribution, the elasticity, and the 3D shape. Our current implementation defines the 3D shape only as a hierarchy of bounding boxes. *AnyElement* does not record any information about external relationships, such as the transformation between an element's reference frame and the ground inertial frame. This information is represented outside the mechanism model in the application that uses it.

AnyPort represents the place where the physical interaction between an element and other elements occurs, such as the mounting points of a link, the working point of an end-effector, the optical centre of a camera, or the contact point of a wheel. *AnyElement* may have any number of *AnyPort* instances, which can represent any kind of physical interaction. Our current implementation defines only the geometric position and orientation of an element's port with respect to the element's reference frame. This allows the computation of forward and inverse kinematics for chains of interconnected elements.

Figure 4 shows an example of *AnyElement* with three *AnyPort* objects P1, P2, and P3 depicted as circles. The arrows represent the rigid transformations between the ports and the element's reference frame R. In order to navigate the geometric relationships between ports efficiently, each port records both direct and inverse transformations. Thus the relative position of two ports, e.g. P1 and P2, is computed as the product of the transformation from P1 to R) and the transformation R to P2. If the element is rigid, these transformations are constant and need to be loaded or computed at initialization time only. Stiffness matrix can be associated to *AnyPort* objects matrices to describe how a vector of external forces applied to an *AnyElement* object cause deformation in *AnyPort* displacement with respect to the *AnyElement* reference frame.

AnyPair represents the interconnection of two elements through their ports. It stores the links to the interconnected ports, the type of the interconnection (e.g. revolute, prismatic, point-contact, etc.) and the value of the articulated position between the two interconnected elements. Typically, *AnyPair* corresponds to a joint between the surfaces of two *AnyElement* objects that keeps them in contact and constraints their relative motion.

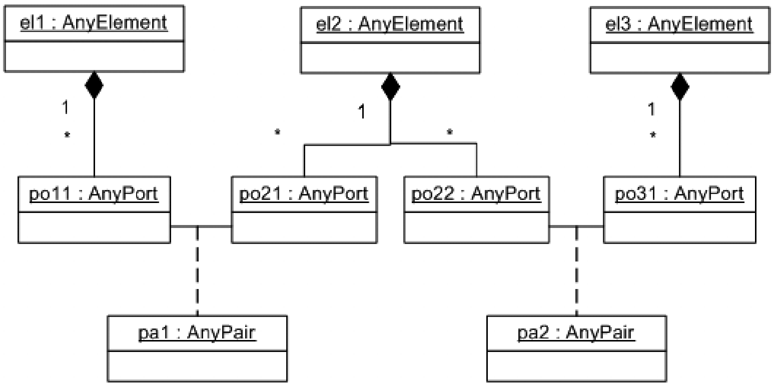

Fig. 5. UML diagram of three interconnected elements

Figure 5 shows the UML diagram that represents the interconnection of elements el1, el2, and el3 through their ports po11 and po21, po22, and po31 by means of pairs pa1 and pa2.

The diagram indicates that class *AnyElement* has a composition relationship with class *AnyPort* , that is, an instance of class *AnyPort* belongs to and is created / destroyed by only an instance of class *AnyElement* . The relationship is bidirectional, thus *AnyPort* objects have a reference to their container object. Class *AnyPair* explicitly represents the association between two *AnyPort* objects, both of which are aware of the relationship between them. As a consequence, when an association between two *AnyPort* objects is created or destroyed, both objects must be informed. If one of the *AnyPort* objects is deleted, the *AnyPair* instance that link them is dissolved as well.

AnyMechanism represents complex mechanisms made up of several *AnyElement* objects, which are concatenated by means of *AnyPort* and *AnyPair* objects. *AnyMechanism* is thus a container software module that encapsulates a set of *AnyElement* objects and maintains the graph of interconnections that make up the mechanism topology. It is in charge of creating and destroying

AnyPair instances and to link/unlink *AnyPort* objects. It embeds algorithms for graph iteration, update, and query.

AnyMechanism supports the hierarchical composition of mechanisms to build more complex mechanisms. For example, a mobile manipulator is the composition of a mobile rover and an articulated arm. In order to allow the definition of algorithms that uniformly iterate the mechanism hierarchical structure, we have applied the Composite Design Pattern [GHJV95]. The result is the data structure depicted as UML diagram in Figure 6, which represents the ANYMORPHOLOGY Design Pattern.

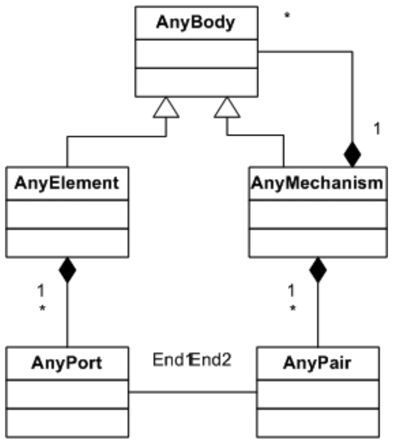

Fig. 6. UML diagram of the AnyMorphology Pattern

The *AnyBody* software abstraction represents both simple *AnyElement* objects and their *AnyMechanism* containers of arbitrary complexity. This is an abstract class that does not need to be instantiated. It implements methods that allow client applications to retrieve the physical properties of individual elements or mechanisms in a uniform way, such as their 3D shape. Thus, clients may even not know if a body is an element or a mechanism.

In turn, *AnyMechanism* objects are made up of an arbitrary number of *AnyBody* objects, which can be *AnyElement* or other *AnyMechanism* objects. *AnyBody* encapsulates the algorithms for iterating the mechanism hierarchical structure.

Figure 7 shows an example of a simplified mobile manipulator structure, where solid line rectangles represent *AnyElement* objects (e.g. chassis, link, . . .), small circles represent *AnyPort* objects, small crosses represent *AnyPair* objects, and dashed line rectangles represent *AnyMechanism* objects. The Robot mechanism encapsulates the Manipulator mechanism and the Rover mechanism. The Manipulator encapsulates three Link elements, while the

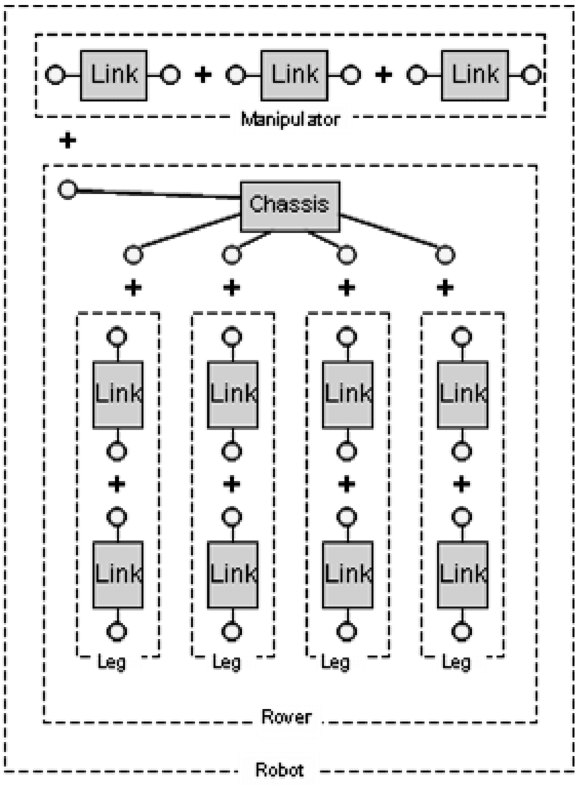

Fig. 7. The topology of a simplified mobile manipulator structure

Rover is made up of the Chassis element and four Leg mechanisms. Finally, the Leg mechanisms are made up of two Link elements each.

The resulting structure can be observed from two distinct perspectives.

First, it represents a flat interconnection of *AnyElement* objects. This structure directly maps the physical chains of mechanical components, which link for example the end-effector of a manipulator arm to the wheels of the mobile rover carrying it. This representation is useful to analyze physical properties of complex mechanisms: an algorithm can iterate through *AnyPort* and *AnyPair* objects in order to compute collision-free relative positions between *AnyElement* objects.

Second, it represents a hierarchical composition of *AnyBody* objects. This structure directly maps the logical encapsulation of partial views on the complete graph, e.g. the Manipulator, the Rover, and the Robot.

This representation is useful to allow a client control application to access mechanical information related to individual mechanisms, e.g. the path planner is concerned only in the rover kinematics.

Generic direct and inverse kinematics and collision detection algorithms are implemented in *AnyMechanism* . Current implementation of kinematics algorithms only work for serial chains. Additionally, *AnyMechanism* objects can be customized with specific algorithms to compute inverse kinematics for specific mechanical structures (e.g redundant or parallel robots [YCLY01]). Customization is done by designing *AnyMechanism* according to the Command Design Pattern [GHJV95]. This pattern defines a way of encapsulating an action and its parameters into an object. This allows adding new operations to existing object structures without modifying those structures.

According to the Command Pattern, the *AnyMechanism* class defines a set of standard interfaces for common algorithms. For examples, forward kinematics algorithms compute the transformation between any two *AnyElement* reference frames. Inverse kinematics algorithms compute the articulation values of a chain of *AnyPort* objects, knowing the transformation between the reference frames of the *AnyElement* endpoints in the chain. Collision detection algorithms compute the distance between the shapes of two *AnyBody* objects. *AnyMechanism* defines the addCommand method that accepts concrete Command objects implementing specific versions of those algorithms. If a new algorithm is defined later, a new Command object is implemented, which can work with *AnyMechanism* automatically.

The Command Pattern allows to encapsulate a single algorithm in one object and to customize a mechanism behavior without the need of implementing a specific adaptation of *AnyMechanism* . Thus, algorithms for complex mechanisms can be implemented taking advantage of the uniform composition of simple mechanisms.

5.3 Software Quality Factors

The ANYMORPHOLOGY Pattern has the following pros and cons. Expressiveness. The proposed model supports the representation of different mechanism topologies, such as the chain of links that make up a serial manipulator, or the tree of devices that make up a mobile rover (transforming chassis, wheel bogies, suspension systems, etc.) or even the graph of serial kinematics chains (legs) that make up a parallel robot.

An interesting consequence of representing the relative positions of *AnyPort* objects explicitly is the possibility to model elastic properties of mechanical elements seamlessly. For example, it is possible to attach a dynamic behaviour to *AnyPort* transformations in order to represent an element deformation due to external or internal forces (e.g thermal dilatation).

Stability. The proposed model clearly separates the invariant properties of a mechanical element from the variable attributes that define how the element is interconnected with other elements to form composite mechanisms.

OROCOS has a similar approach, but the ANYMORPHOLOGY model keeps the representation of the element interconnections as simple as possible. This makes the model resilient to changes in the application requirements specifications, as it supports the representation of new mechanism topologies seamlessly, such as in the case of reconfigurable robots.

Scalability. The model is highly scalable thanks to its modular structure. Every mechanism is modelled as a separate entity, whose properties are defined within an individual module. AnyMecahnism objects encapsulate the constituent elements, their interconnections, and the algorithms that manage them. Complex *AnyMechanism* objects are assembled from building-blocks mechanisms that completely hide their internal structure.

Efficiency. The proposed model supports efficient traversal of complex mechanical structures thanks to the hierarchical organization of views on the flat elements graph. Even if the graph size is very large, search and update algorithms can exploit the mechanism composition hierarchy to retrieve information on every element.

The design choice of representing geometric transformations as *AnyPort* objects limits the performance of kinematics algorithms since every transformation between two elements' reference frames is represented as the composition of two *AnyPort* transformations. Nevertheless, most of the typical mechanical elements used to build robots (such as manipulator links) are represented by two-ports *AnyElement* objects. The reference frame of these elements is usually set in correspondence of either of their ports, thus, they relative position is represented by a single transformation.

5.4 How to Use the ANYMORPHOLOGY Mechanism Model

As stated in Section 5, the goal of the ANYMORPHOLOGY Pattern is to support the development of software control applications that are reusable across different robotic platforms, by making software dependencies to the robot mechanical structure explicit.

This means, that typical functionality such as motion control, path planning, and navigation are implemented on top of the mechanism model by using its algorithms to retrieve and compute kinematics information. If the mechanical structure changes, the mechanism model and its algorithms need to be update, but the implementation of the robot functionality is not affected.

Four alternative usage scenarios demonstrate the flexibility of the proposed model. In the first scenario, a copy of the mechanism model is instantiated for each software functional component. The model records the current configuration state of the robotic mechanism (i.e. the values of the *AnyPair* articulations), while the functional component is in charge of keeping it up-to-date (e.g. by reading the robot sensor). This scenario allows the component to optimize the update rate and to have exclusive access to the model state. Components might even maintain only a partial view of the mechanism state (e.g. only the rover state). The main disadvantage is the waste of memory to load several copies of the same mechanism model and the difficulty of synchronizing different components.

In the second scenario, a single instance of the mechanism model acts as a central repository of the mechanism state for all functional modules. It is in charge of managing the synchronized access to the state by different components. The main disadvantage is the limited performance due to the concurrent access to the central repository.

In the third scenario (similar to the CLARAty approach), a central repository maintains only information related to the invariant properties of a robot mechanism, i.e. the values of *AnyElement* and *AnyPort* attributes. Every functional component that uses the model is in charge of providing a the mechanism state information, i.e. the values of *AnyPair* attributes. The current state is passed to the methods of the mechanism model every time an algorithm has to be executed. The main advantage is that multiple clients can share the same mechanism model efficiently. This is also powerful for reconfigurable robots where a single copy of the robot topologies is maintained. The main disadvantage is that every client (i.e. functional component) has to manage and provide mechanism state information. Different components may share the mechanism state if they agree on a common representation.

In the fourth scenario, not yet supported by the current implementation, the central repository instantiates as many copy of the mechanism state as needed. The idea is to extend the design of the ANYMORPHOLOGY Pattern according to the Iterator Design Pattern [GHJV95], which allows a way to traverse the elements of a composite object without exposing its internal structure. Every functional component holds an instance of an Iterator object, which stores the current state of the robotic mechanism. The component is in charge of updating the Iterator image of the mechanism state. Iterator objects are returned by the mechanism model on demand and can be easily cloned in order to share the same mechanism state with other functional components.

Another intriguing aspect related to how to use the ANYMORPHOLOGY Pattern is the relationship between the model of a mechanism and the external inertial reference frame. It is clear that the model records the state of a mechanism only in configuration space, i.e. the values of the *AnyPair* articulations. Once the transformation between the reference frame of at least one mechanism element and the inertial frame is known, the state of any mechanism element in Cartesian space can be derived from the mechanism state by computing the direct kinematics.

A question rises on how to represent the relationship with the inertial frame. The first solution requires the client functional components (e.g. the navigator) to record the global positions of any of the mechanism elements and to compute the positions of the other ones using the mechanism model internal configuration state and the kinematics algorithms.

The second solution, which actually does not adhere to the overall stability-oriented approach of the ANYMORPHOLOGY Pattern, consists in adding a "virtual" *AnyPort* objects to every *AnyElement* object to represent the transformation between its reference frame and the global inertial frame.

6 Conclusions

The three EBTs presented in this chapter and the ANYMORPHOLOGY Pattern have been applied to model different type of robotic systems. In particular, a Kuka KR16 industrial manipulator, a Pioneer 3DX rover, a Labmate mobile platform, and several simulated robots. Actually, the benefits of appling the proposed Software Patterns can be appreciated more from a software development perspective. It simplifies the development of new control applications and the reuse of an application on different robotics platforms.

References

[Ark98] R.C. Arkin, *Behavior-based robotics (intelligent robotics and autonomous agents)*, The MIT Press, 1998.

[Bro91] R.A. Brooks, *Intelligence without reason*, in Proc. of the 12th Int. Joint Conference on Artificial Intelligence (1991).

[CG00] M. Clien and M. Girou, *Enduring business themes*, Communications of the ACM **43** (2000), no. 5, 101–106.

[CHF05] Y. Chen, H.S. Hamza, and M.E. Fayad, *A framework for developing design models with analysis and design patterns*, The IEEE Int. Conference on Information Reuse and Integration IEEE IRI-2005, Las Vegas, 2005.

[CHW98] J. Coplien, D. Hoffman, and D. Weiss, *Commonality and variability in software engineering*, IEEE Software **15** (1998), no. 6.

[CZ00] R. Chrisley and T. Ziemke, *Embodiment*, Encyclopedia of Cognitive Sciences (2000).

[FA01] M.E. Fayad and A. Altman, *An introduction to software stability*, Communications of the ACM **44** (2001), no. 9.

[Fay02] M.E. Fayad, *Accomplishing software stability*, Communications of the ACM **45** (2002), no. 1.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of reusable object oriented software*, Addison-Wesley, NY, 1995.

[KDR04] K. Kawamura, W. Dodd, and P. Ratanaswasd, *Robotic body-mind integration: next grand challenge in robotics*, 13th IEEE Int. Workshop on Robot and Human Interactive Communication (ROMAN2004), September 2004.

[Mat02] M. Mataric, *Situated robotics*, Encyclopedia of Cognitive Sciences (2002).

[Sim69] H. A. Simon, *The architecture of complexity*, The Sciences of the Artificial (1969), 192–229.

[BR05] D. Brugali and M. Reggiani, *Software Stability in the Robotics Domain: Issues and Challenges.* In Proceedings of the IEEE International Conference on Information Integration and Reuse (IRI2005), Las Vegas, Nevada, August 2005

[BS06] D. Brugali and P. Salvaneschi, *Stable Aspects in Robot Software Development*, Journal on Advanced Robotic Systems, Vol.3, N.1, March 2006, pp. 17-22.

[OSCAR] *The OSCAR Project.* Online resources at http://www.robotics.utexas.edu/rrg/research/oscarv.2/

[OROCOS] *The OROCOS Project.* Online resources at http://www.orocos.org

[Nesnas06] I.A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, *CLARAty: Challenges and Steps Toward Reusable Robotic Software,* International Journal of Advanced Robotic Systems, Vol. 3, No. 1, pp. 023-030, 2006.

[Otter96] M. Otter, H. Elmqvist, F.E. Cellier, *Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola*, Nonlinear Dynamics, Vol. 9, pp. 91-112, 1996

[SPK00] R. Sinha, C.J.J. Paredis, and P.K. Khosla, "Integration of Mechanical CAD and Behavioral Modeling," IEEE/ACM Workshop on Behavioral Modeling and Simulation, Orlando, FL, USA, 2000.

[PTKT02] Mitchell W. Pryor, Ross C. Taylor, Chetan Kapoor, and Delbert Tesar, *Generalized Software Components for Reconfiguring Hyper-Redundant Manipulators*, IEEE/ASME Transactions on Mechatronics, Vol. 7, No. 4, Dec. 2002, pp. 475-478

[RML] *The RoboML Project.* Online resources available at: http://www.roboml.org/

[BSK03] H. Bruyninckx, P. Soetens, and B. Koninckx, *The Real-Time Motion Control Core of the Orocos Project*, In Proceedings of the 2003 IEEE International Conference on Robotics & Automation, pages 797-802, Taipei, Taiwan, 2003.

[NWBS03] A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, Won Soo Kim, *CLARAty: An Architecture for Reusable Robotic Software,* SPIE Aerosense Conference, Orlando, , April 2003.

[CLR] *The CLARAty project.* Online resources at http://claraty.jpl.nasa.gov

[DNNK06] A. Diaz-Calderon, I. A. D. Nesnas, H. Das Nayar, and W. S. Kim, *Towards a Unified Representation of Mechanisms for Robotic Control Software*, Journal on Advanced Robotic Systems, Vol.3, N.1, March 2006, pp. 61-66.

[DDS] *The DARTS/DShell Project.* Online resources at http://dartslab.jpl.nasa.gov/ROAMS/index.php

[Jain91] A. Jain. *Unified Formulation of Dynamics for Serial Rigid Multibody Systems*, Journal of Guidance, Control and Dynamics, vol. 14, 1991, pp. 531-542.

[YCLY01] Guilin Yang, I-Ming Chen, Wee Kiat Lim, Song Huat Yeo, *Self-calibration of three-legged modular reconfigurable parallel robots based on leg-end distance errors*, Robotica (2001) volume 19, pp. 187-198. Cambridge University Press