# Survey on Architecures and Frameworks for Autonomous Robots

Piotr Makowski
(makowski cs.aau.dk)

November 7, 2004

## 1 Historical background

The first both architecturally documented and empirically tested autonomous mobile robot, named Shakey [86], was presented in the late 1960s by Stanford Research Institute (now SRI International). That robot was intended as a combined demonstration of achievements in the three various, independently evolving areas of Artificial Intelligence: perception, locomotion and problem solving. Such a division, strongly embedded in philosophical trends of AI development [85] was a foundation for the first autonomous robot architecture, known in the literature as *SPA (sense-plan-act)* or *deliberative*. The term 'deliberative' originated from the mostly exposed problem solving layer that contained exact symbolic representation for an external environment along with reasoning mechanisms for it. Although deliberative systems left a certain imprint on robotic development outlining most of the problems and challenges in the new research area, they were strongly criticized [15, 16], among other things for their slow responsiveness and a strong dependence on a highly elaborated laboratory environment. Indeed, environment modeling and planning appeared to be highly complex computational problems [21].

In the midst 80s Brooks [13] came up with a famous alternative for mobile robots known as *a subsumption*, or (in opposition to deliberative) *reactive* architecture. The architecture tightly coupled a decisive layer, represented by a hardwired hierarchical set of *augmented finite state machines* (AFSM) to the robots sensors and actuators. A finite state machine responsible for a specific behavior could suppress or inhibit the other depending on the observed context. Reactive robots appeared, in comparison with deliberative, to have dramatically improved performance. However experimental robots

Figure 1: The robot Shakey

[26, 27, 15] constructed in the following years (such as an office robot Herbert presented by Connell, or a Cog project announced by Brooks as new step of subsumption architecture that between 1993 and 2003 completely changed its direction) revealed that it is extremely hard to extend subsumption based robots with new higher level behaviors. A lack of modularity [50] hampered development of reactive robots before they reached practical application level. As a solution to this problem various groups of researchers started looking at the idea of combining both deliberative and reactive approaches within a single system. In 1989 Arkin [5] provided the concept of a deliberative/reactive hybrid architecture applying both psychological and neuroscientific models for integration. He introduced the *Autonomous Robot Architecture* (AuRA) framework [6] as an experimental test-bed for various environmental settings. Around 1991 several groups of independent researchers [28, 43, 10] came up with another architectural solution. They synthesized reactive control and deliberative layers introducing a sequencing mechanism as a mediator between them. The approach is widely known as the *three layer* architecture [46]. It is applied in many recent robotic experiments [44, 11, 36]. Another approach for extending reactive systems with higher level behaviors, called *motivation networks* [61, 103], took its inspirations from biological sciences. It combines reactive feedback control mechanisms with a motivation subsystem responsible for selecting and modulating robot behaviors. Today, although even simple robots [23] built with the motivation principle show impressive diversity of interesting behaviors, it still remains unclear how to design such a systems to provide expected and repeatable results.

The concept of forming a group of robots acting upon completion of a single task appeared in early 90's [75, 22, 88] promoted by the results in a distributed problem solving and studies on various biological societies like

ants, bees or birds. A *multirobotic* team can potentially be more robust and efficient, exploiting natural parallelisms in a task execution. Also technical requirements for group of heterogeneous, specialized robots seem less demanding than for a construction of a single multipurpose robot. However, multirobotics solutions have given a number of new challenges related to e.g., multirobotic motion planning or rules for group organization and interaction. Recently developed multirobot architectures show a number of options in applying collective strategies and draw their inspirations from many different sciences ranging from theoretical biology [89] to game theory [7], sociology or economics [32].

As autonomous solutions grew in complexity, their development became a challenging programming activity per se. As a result in the late 80s a number of programming languages dedicated for control units in reactive systems were released. They were mostly offering a high level language structures for a control flow management or data structures and primitives dedicated to a particular robotic architecture. Around 2000, the robotic community started looking into so called *frameworks* as means of providing more sophisticated support for development including hardware abstraction layer (HAL), communication protocols and dedicated modeling languages.

# 2 Systems Architecture Principles

Development of a mobile autonomous robot comprises activities in both control engineering and computer science. The autonomous robot architecture can be seen both as a control feedback system, reacting to the continuous or discrete changes in the environment in a timely manner and as a reactive system performing computations on sensory produced values and returning results to actuators. Both definitions seem too broad, fitting to the autonomous robots architecture as well as to any other robotic system. In order to gain a deeper insight into autonomous robotic area we need to outline the philosophy and principles of autonomous mobile robots design. In the following sections we shall briefly describe the most popular robotic solutions focusing on the main architectural aspects and the ways they interact to provide expected results. The next section (2.1) describes the most popular single robotic architectures. Section 2.2 discusses possible approaches for constructing multirobotic teams. The section 2.3 addresses important extension of robotic architectures - adaptivity and machine learning.

## 2.1 Single Robot Architectures

Considering progress in AI over the last several decades, one notices a qualitative revolution in the approach to the principle aspects of human thinking and intelligence. Starting from a concept of axiomatic theories, and the invention of knowledge bases manipulated with methods of natural deduction[1], we see a deeper understanding of the complexities in non-monotonic reasoning, the significance of the frame problem and inherent limitations of various expert systems. Concurrently we observe a number of attempts at expressing intelligence as the capability to learn and adapt to an arbitrary nonlinear function. The suggested methods, usually biologically inspired, such as evolutionary algorithms or neural networks provide insight into the role of generalization and abstraction mechanisms. Development of autonomous single-robot architectures completes the picture of "intelligence oriented research" opening a number of questions on the nature of deliberation and reactivity in time-constrained environments, and the level of intelligence that can arise from sets of a simple rules.

The previous chapter outlined these several competing concepts of robotic autonomy. In the following sections, the most popular architectures shall be briefly discussed, namely: reactive, deliberative, three layer and motivation based. The list is by no means exhaustive of the original concepts appearing in the literature. The choice of whether some architecture deserves a separate chapter or should be classified as a special case of another is always subjective and arguable. To the best of our knowledge, the presented selection is representative for the current trends in robotic development. However there are other architectures that could have been abstracted into separate chapters. For example, Nilsson [87] presents *teleo-reactive programs* encoding reactive control into a sequence of closures. We consider that approach a realization of a general reactive architecture (Section 2.1.2) rather than a separate design concept. Maes [66] describes an interesting autonomous control mechanism based on *activation spreading*. It can be seen as an example of motivation networks presented in section 2.1.4. The *DD&P* (The dual Dynamics and Planning) architecture [94] resembles the one described by Maes. Yet it contains a number of original concepts such as *qualitive activation* patterns, why we mention it here. There is a great number of the other architectural examples that could constitute additional sections in this chapter, but time and space limit us in the end.

[1]The idea was popularized much earlier by a famous character from Conan Doyle novels.

### 2.1.1 Deliberative Architecture

Probably one of the most crucial aspects influencing a conceptual model of an autonomous robot is the meaning given to the word "autonomy". Designers of deliberative architectures associated autonomy with intelligence, adopting an interpretation commonly accepted in the community of AI researchers. Accordingly "autonomous robot" means a robot acting like a human (cognitive model) or robot acting rationally, i.e. actions can be strictly deduced from a formal model. Due to the similarity to human behaviors, the deliberative architecture (Figure 2) is probably the most intuitive one, having a cleanly defined relation between components (architectural parts) and their responsibilities.

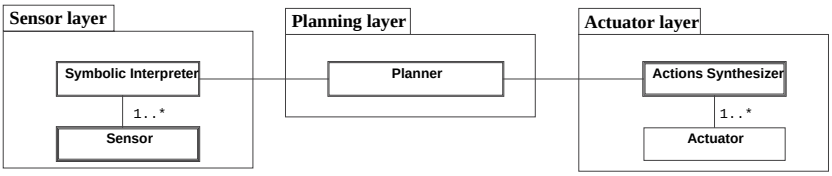


Figure 2: Deliberative architecture

The deliberative robot, Shackey, was acting in a specially built laboratory area, where several different blocks were stored. The task of the robot was to move these blocks from one room to another according to some general specification. The task execution was controlled by a deliberative layer based on the STRIPS [37, 79, 80, 62] planning system. The STRIPS planner manipulates an environment model given in a subset of first order predicate logic. For example propositions `IN(A,R1)` or `ON(A,B)` would be used in STRIPS system to denote facts like: *"block A is in room R1"* or *"block A is laying on block B"*. STRIPS applies a *closed world assumption* for facts representation, i.e. only propositions that do not contain variables are allowed. The operations are specified in terms of PDA (pre-condition, delete, add). Given that a certain precondition holds, the operation can be applied resulting in deletion (addition) of propositions in the environment model. For example, the operation *MOVE(A,B,C)* that removes block A from block B and puts in on block C could be executed, assuming that block A lays on block B (`ON(A,B)`), resulting in deletion of the `ON(A,B)` proposition and addition of `ON(A,C)`.

Formally the planing problem in STRIPS can be formulated as follows: Given a set $A$ of operator specifications, an initial set $\Sigma$ of known propositions and a set $\Omega$ of goal propositions we look for a sequence of operations (a plan) leading from set $\Sigma$ to a set containing $\Omega$. Considering that all prepositions

in STRIPS are closed we can reformulate the planning problem as: Given a tree of an infinite depth where each edge in the tree is labeled with some operation from $A$, then label every node with a set of prepositions in such a way that the preposition set for any node $\phi$ can be strictly derived from its parent preposition set by applying the operation assigned to the edge between them. Assuming that the root of the tree is labeled with $\Sigma$ we are looking for a path of the shortest length, leading from the root to some node labeled with a superset of $\Omega$.

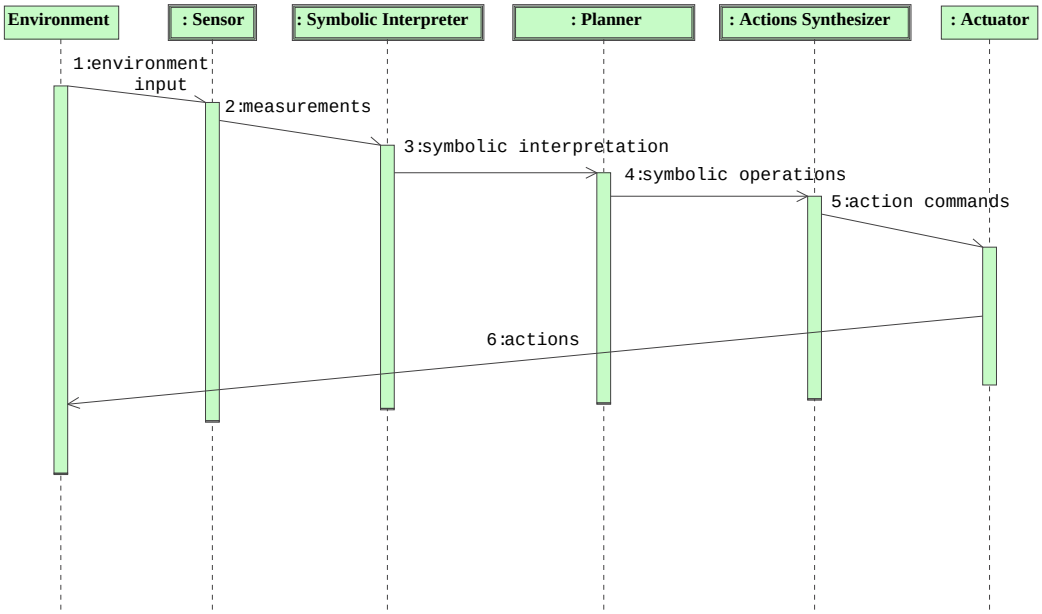


Figure 3: Typical cycle of execution in deliberative architecture.

Such a processing model provides general contracts for the Sensor and Actuator layer (see Figure 3). The Sensor subsystem provides a set of known facts which are interpreted as prepositions (eg. `ON(A,B)`, `IN(B,R1)`). The Actuators subsystem is responsible for symbolic operators execution (eg. `MOVE(A,B,C)`, `CLEAR(B)`) resulting in a physical changes in the environment.

There are several aspects contributing to disappointing performance of deliberative architectures:

- The robot world model is incomplete or inaccurate. (For example the doors behind the corner the robot plans to pass through are now closed.)
- The world is changing spontaneously. (Someone is pushing block `A` left and right.)

- The robot actuators provide different results from the expected ones. (The robot fails to put block `A` on `B`. It slips down.)
- The sensors measurements are inaccurate or incorrectly interpreted. (There are two blocks `A` in the room.)

All these commonly faced situations are reflected in a deliberative system by a constant flow of new propositions (that are often inconsistent with the actual model). As a result the system frequently performs model updates (adding new prepositions, discarding inconsistent ones), invalidates existing execution plans, and restarts the planning procedure using the updated model.

Recently a new generation of prepositional planners appeared, providing more efficient solutions to deliberation problems. They usually exploit various heuristics such as a *bounded horizon* assumption, a hierarchical task decomposition or a probabilistic search model. Nau et al. [84] present HTN (Hierarchical Task Network)[2] planner called SHOP (Simple Hierarchical Ordered Planner), that allows a dynamic plan correction in case of unexecutable actions. It uses backtracking method to find the lowewst level of decomposition, where an alternative is available. PDDL (Planning Domain Description Language) [81] is a new expressive language (inspired by STRIPS syntax) for which a number of a planner implementations is widely available. Fox and Long [40] define an extension to the PDDL subset that allows formulation of temporal properties in the plan.

### 2.1.2 Reactive Architecture

Designers of an reactive architecture assigned a completely different interpretation to the word "autonomy". Their focus was primarily set on the robot's ability to act independently in the real world environment. Quoting from Brooks [17], central ideas behind reactive approach were:

> Situatedness: The robots are situated in the world—they do not deal with abstract descriptions, but with the "here" and "now"

[2] "In HTN planning, the planning system begins with an initial state-of-the-world and with the objective of creating a plan to perform a set of tasks. HTN planning is done by problem reduction: the planner recursively decomposes tasks into subtasks, stopping when it reaches primitive tasks that can be performed directly by planning operators. In order to tell the planner how to decompose nonprimitive tasks into subtasks, it needs to have a set of methods, where each method is a schema for decomposing a particular kind of task into a set of subtasks (provided that some set of preconditions is satisfied). Unlike classical planning, HTN planning is Turing-complete." - Provided after SHOP project description.

of the environment that directly influences the behavior of the system.

Embodiment: The robots have bodies and experience the world directly—their actions are part of a dynamic with the world, and the actions have immediate feedback on the robots' own sensations.

The desired "autonomy as intelligence" was believed to appear when new higher level behaviors were added to simple and robust reactive robots. That belief was based on similarities to Darwin's evolution theory, stating that more sophisticated skills appear as small step extensions to simpler systems.

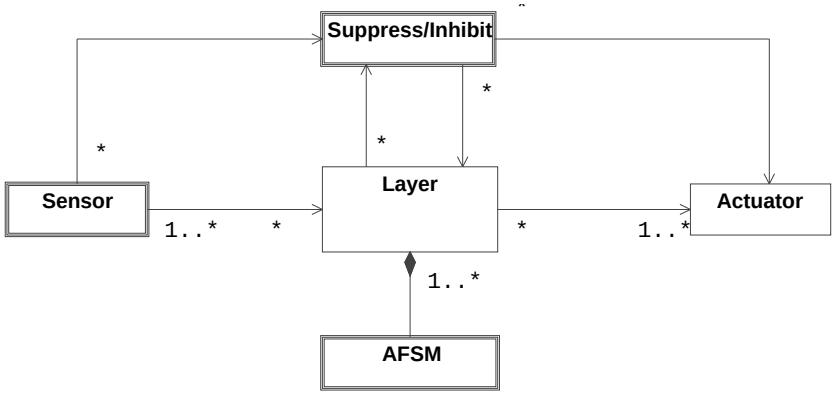


Figure 4: Reactive architecture (Class Diagram)

Reactive architecture (see Figure 4) admits almost arbitrary relations between comprizing elements. The original architectural concept (sometimes referred to as the *Subsumption Architecture*), presented by Brooks [13, 14], structures a reactive system into a set of strictly ordered layers (see Figure 5). Each layer is responsible for a single behavior (the more complex behavior, the higher the level). A layer can read sensor measurements and react according to its competences via actuators. Higher layers can suppress/inhibit lower ones, explicitly taking control over robot reactions. A layer contains a set of communicating Augmented Finite State Machines. The AFSM is a classical timed automaton extended with a set of registers and timers. The registers can be set by signals from sensors or other AFSMs. Usually each AFSM has given its own thread of control (e.g. each AFSM has assigned a simple processor). It allows different parts of the system to react independently from one another.

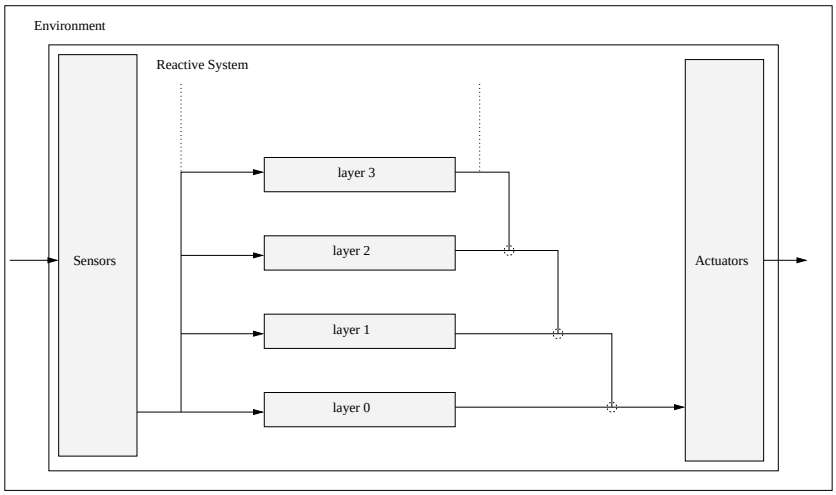


Figure 5: Reactive architecture

Use of AFSM as building blocks for reactive robots resulted in short and bounded times. Over the following years several groups of researchers provided various experimental robots demonstrating robustness of reactivity. In 1987 Jonathan Connell's simple Tom and Jerry robots [26](equipped with a single 256 gate programmable array logic chip) were able to avoid obstacles and follow moving objects. The robot Herbert [15], presented by Rodney Brooks in 1990, was wandering around the office room, avoiding obstacles and stealing empty soda cans from offices desks. The robot was equipped with twenty four 8-bit CMOS microprocessors combined into a single distributed computer. Another robot TOTO, presented by Maja Matarić [72, 76] between 90 an 92, was capable of navigating in an office-like environment and building its own topological maps of the environment. TOTO was processing all the information with a single Hitachi CMOS 68000 microprocessor.

The growing complexity of robotic capabilities revealed that the theoretically simple and structured reactive architecture in practice loses its modularity leading to monolithic, highly complex design like the one presented after Brooks [14] on Figure 6. Even considering programmable controllers, where the reactive achitecture is implemented rather as a set of programs than a bunch of thightly connected wires, the scalability remains an inherent problem [70]. As the main reasons for diminishing hierarchy one may point at:

- *ordering problems* - hierarchy based on an ordering like "more complex" or "higher level behavior" is not well-defined.

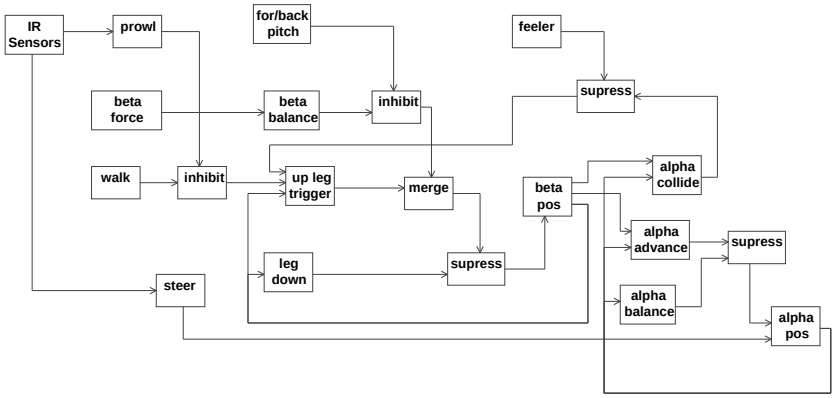


Figure 6: Reactive architecture - realistic view

- *competence clashes* - in principle layer A can suppress layer B and vice versa, depending on a larger (including previous events) context.
- *extension methodology* - a methodology for adding new layers in a way that would preserve system dynamics is not defined.
- *scarce internal state representation* - modeling an autonomous robot as a finite state automaton, can lead to situations where actions taken by a robot does not lead to any changes in the environment. Finite state automaton would endlessly repeat a sequence of actions unless externally it appears as hopeless (i.e. it would not accept anything but regular languages as protocols for environment).
- *initialization method* - as a reactive robot reflects the state of an environment merely through states of its AFSM's, a proper initialization requires extending each AFSM with an extra initial state and a number of transitions to other states reflecting possible initial environmental conditions.

### 2.1.3 Three layer architecture

The deliberative and reactive architectures represent the opposite "extremes" in the approach to the robotic design. The first is aiming at the perfect internal representation for an external environment, the second at the maximal adaptation with possibly no internal state at all (*"the world as its own model"*). While the experimental robots based on these two approaches in general are incomparable, showing the different and usually complementary sets of the skills, the effective combination of these two appears as a promising

solution. The similar architectures (see Figure 7) that integrated both deliberative and reactive systems appeared in the three independent publications under different names: *SSS* (J. Connell) [28], *3T* (R. Bonasso)[10], *Three-Layer Architecture* (E. Gat)[46]. All these architectures related deliberative and reactive subsystems according to their timing and complexity capabilities. While each of the presented solutions contains a number of unique and author specific thoughts, they all contribute to the same approach of robotic development providing different views on the same architectural concept[3].

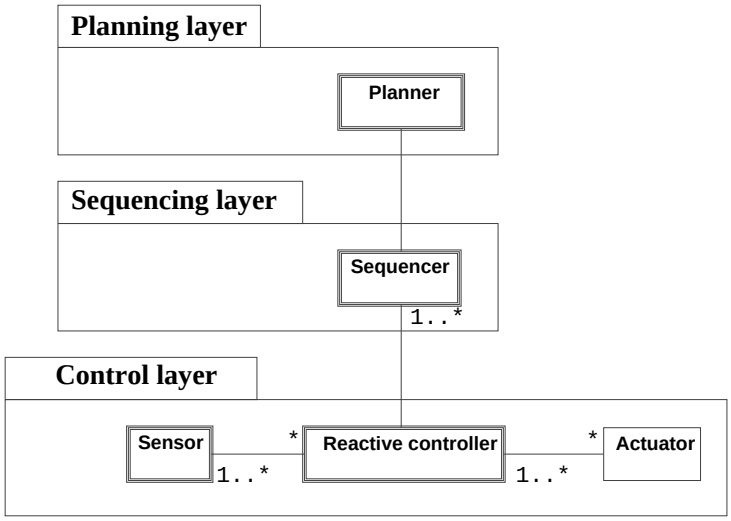


Figure 7: Three Layer Architecture

The *control layer* (lowest level) in the architecture is responsible for real-time monitoring and control over environmental changes. It consists of a number of sensors and actuators connected by one or more programmable controllers into a closed feedback loop. The control program simulates set of finite state machines that play the same role as AFSM's in reactive systems. Instead of extending the program with new state machines to represent additional behaviors, designers of three layer architecture try to keep it as simple as possible. It addresses only a very restricted context (e.g. a single primitive behavior). Whenever the control program fails in goal achievement or becomes inadequate to the recent situation, it is replaced with another one uploaded (see Figure 8) by a *sequencing layer*. The sequencing layer contains a selection of various control programs structured according to their behavioral roles and related to one another as alternatives. The sequencing

[3]The name *"Three layer architecture"* is thus adopted after the number of layers in each solution rather then after its original appearance.

mechanism is able to translate deliberative high level plans into the lists of control programs downloaded to the controller one by one. It can also react to the direct request from the control layer, conditionally formulating alternative sequence of control programs. The concept of uploading alternative control programs *"on demand"* provides efficient solution to some problems common in reactive systems like initialization or scarce internal state representation problems (see section 2.1.2). The *planning layer* contains a deliberative STRIPS-like planner. It can also be extended with a user interface and some low speed or computational intensive sensors (like vision systems) that provides facts for the deliberative planner. The planner can interact with the sequencing layer as either a higher level controller or an expert system. In the first case it is responsible for providing sequences of operations for an execution. In the second case it can reply to the particular queries computing alternative plans or providing high level facts from the model.

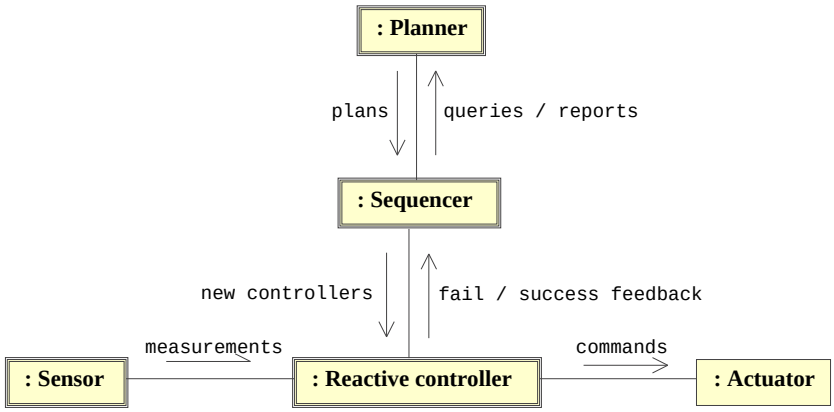


Figure 8: Three Layer Architecture - communication between layers

The three layer architectures owe partially their empirical success to the significant progress in the manufacture of an electronic equipment. The high availability of fast programmable controllers, and wireless connectivity devices allowed a larger flexibility in physical implementation, making technical decisions less important. Among the many robotic examples (TJ [28], Homer, Robby [46], Chip [11], Diligent [36]) demonstrating various flavors of three layer architecture, only few mention physical properties of an equipment involved. Instead one could observe an explosion in the number of new programming languages and formalisms supporting development of robotic systems on various levels of abstraction. Some of the documented examples are *Behavior Language* [17], *ALFA* (A Language For Action) [42] supporting development of reactive parts of the system, *PRS* (Procedural Reasoning system) [47], (*RAP* (Reactive Action Package) [38], REX/GAPPS [59], ESL

[45], TDL (Task Description Language) [95] dedicated for a task management and sequentializing process.

### 2.1.4 Motivation Network

The motivation network is another approach for extending reactive systems with higher level behaviors. It adapts biological observations on animal behaviors to establish a model of a robot autonomy. Quoting Krink [61]:

> The behavior of animals is based on motivation and decision making in respect to their generic background and individual life history. Constantly shifting priorities lead the animal to "decide" how to act in particular situation.

Conceptually similar architectures (see Figure 9) for motivational networks provided in [61] and [103] show some similarities to the Three Layer Architecture.

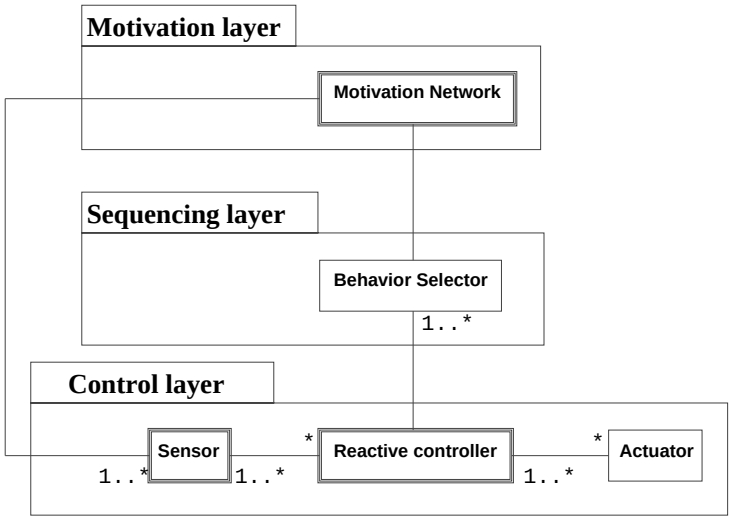


Figure 9: Motivation Network

In both cases the lowest level contains a reactive controller running a program that addresses a restricted context (e.g. representing a single primitive behavior). The upper level contains a selection of controller programs (primitive behaviors) organized according to the higher level behaviors. The main difference in comparison with the Three Layer Architecture is a strategy for selecting higher order behaviors. Both sequencing mechanisms used on the tactical level and deliberative planner on the strategical one are replaced by a so called motivation network. The Motivation Network models

an "animal-like" internal state as a set of variables that intuitively represent robot motivations (e.g. lonely, exhausted, confused). These variables are dynamically updated according to the environmental changes measured by sensors (called stimuli) and some notion of a passing time (circadian clock). Each motivation variable has statically assigned a characteristic behavior encoded as a sequence of primitive behaviors. The values of motivation variables are constantly mapped into a linearly ordered set (usually into some interval of real numbers). The decision on the behavior that should gain priority and be selected next is based on comparison of mappings (e.g. the higher value the higher motivation).

The concept of autonomy and intelligence in motivation networks differs significantly from the ideas appearing in both deliberative and reactive architectures. Instead of providing an exact model for an environment (either as a symbolic representation or as a thoroughly elaborated internal structure) and methods for the interaction within, motivation network focuses on a model of an individual "being" reacting not only according to the external context but also to its own internal state and history. Motivation based robots does not need to cope with the problem of maintaining a constant synchronization with the environment. Experimental examples [23] shows that initialization and scarce internal representation problems (see. 2.1.2) are also non-issues. Constant change of dominating motivations according to the internal circadian clock, guarantees that a robot will eventually select some appropriate behavior for its context. Additionally the designers of motivation network postulate that it is possible to use a robot's successes and failures in goal achievement in order to dynamically modify the internal structure of the network. Such a mechanism, simulating learning by experience (reinforcement learning) would effectively increase robot robustness and adaptiveness over time.

The simplicity and potential richness of the motivation based solution comes at a certain price: A lack of a strict relation between an environmental context and an observed robot behavior makes debugging and modifying of a robot (e.g. removal of unwanted behaviors) a serious challenge. Effective methods for extending robots of these kind with new behaviors are also not known currently.

## 2.2 Multirobot Architectures

The term - *multirobot solution* is usually used in a reference to a collection of robots which participate actively in a single mission or whose actions deterministically lead to the same "global" result. There are several applications in which multirobot teams appear superior to the single robot based solu-

tions.

- Some tasks are too complex for a single robot. For example a task that require simultaneous manipulation from several points in space or includes complex operations on a single object.
- Multirobot teams can perform many tasks at a time exploiting parallelisms in plans. Such an approach can be particularly efficient for the missions that embrace many loosely related activities.
- Specialized, dedicated robots are cheaper, easier to construct and more flexible than multipurpose single robotic solutions.
- Tasks like harvesting, systematic "search and rescue" missions, foraging (picking up and gathering scattered objects as a simulation of a toxic waste cleanup) are naturally distributed.
- Multirobot architectures are also considered as interesting experimental testbeds in the research of organizational theories.

In context of autonomous robots, where each robot forms an independent entity, the successful task achievement requires some higher level models, called *collective behaviors*, describing how the single robot acts within a group and how it influences the team organization and performance. Collective behaviors are usually associated with design decisions such as: How do the robots model the other group participants? Are they aware of the tasks constituting the mission they are working on? Do they communicate with the other group members? What are the subjects for a negotiation? They have a large impact on distribution of competences in the group, modeling system dynamics and leveraging complexity of the solutions. Currently, multirobotic literature mentions four main "streams" of the collective behaviors. However various combinations and variations are quite common.

The first model - *Collaborative behavior* assumes that the robots are aware of the common mission and share the same goals. However they are acting largely independently over the assigned tasks. The possible synchronization is held on a level of planned tasks and used for agreeing on the task order and allocation.

*Cooperative behavior* is a model in which each autonomous robot is aware of the team members abilities and can act synchronously with the others on performing a single subtask. That approach usually requires intensive communication (either direct or indirect) between the group members. A typical example of cooperative behavior is a group of robots precisely manipulating a single heavy object [97], where one of the robots is used for lifting and

caring and the other for adjusting the object position. Another possibility for cooperation is to synchronize efforts without the use of a direct communication. As an example two robots can push/drag the same object to a given position, using it as means of communication [24] i.e. each robot reacts to the changes in object position caused by the other robot actions forming dynamic feedback loop. The first two models usually relay its efficiency on the assumption that robots act unselfishly. They will choose to share the single task whenever it may lead to the increased performance. Alternatively subtasks are distributed apriori between the group members according to their capabilities and other robots react to the requests for help.

In a *competitive* model each robot is individually rewarded for a subtask completion. It is aware of subtasks available within a task and can compete with each other in order to maximize its reward. That solution usually involves some arbiter mechanisms (e.g. election protocol) and additional rules for assigning subtasks and assessing the robot performance [32].

*Emergent behaviors* originated from a phenomena observed in societies of biological creatures such as ants or birds. According to it, individual behaviors constrained by simple local rules can lead to intractable centrally but deterministic [92] group behaviors. A classical example is a bird flocking and its simulations [93] in which simple individual rules of following general direction and maintaining distance to the "visible" neighbor resulted in complex and highly stable group formations.

Collective behaviors, useful for describing overall group interactions, are not sufficient for providing full description of a multirobotic system. Although some architectural solutions seem to be more suited for the particular model, the actual choices are not determined and can be considered independently from the group behaviors. The technical issues commonly discussed in context of multirobotic design include:

**Centralized vs. Decentralized.** The execution control which can be either *centralized* or *decentralized*. The centralized architectures contain a single unit (robot, planner, operator) for controlling a job execution. In decentralized ones the control is usually divided between the group members. It can be fully *distributed* giving all the robots equal responsibilities in controlling a progress or *hierarchical* if there are several "leader" robots capable of controlling the other. The centralized architectures are found less reliable due to a single point of failure. Also the performance requirements for a control unit grow rapidly with the team size. On the other hand, fault tolerance and error recovery (e.g. replanning) capabilities in distributed architectures are much more difficult to implement due to a lack of a global state notion. As most of

the recent experiments with a distributed control use small groups of robots, it is also hard to predict scalability of the approaches of that kind.

**Homogeneity and heterogeneity.** The teams can be formed by a number of identical individuals or by specialized robots providing different capabilities. In general heterogeneous architectures are more difficult to design. For example different robot capabilities need to be explicitly considered during the planning and the task assignment. Also communication protocols become non-symmetric and more abstract as they carry various - robot specific contents. The significant advantage of heterogeneity is that it allows to reduce the number of different devices that needs to be installed and synchronized on a single robot. Robot construction can be thus simplified and tuned to support planned applications.

**Synchronization.** The team members can synchronize *directly* using some wireless communication protocol. For example they can exchange the information about the job status, provide sensory measurements in a timely fashion or coordinate activities upon a shared task. Another option is to synchronize the robots *via the environment*. In this case the robots are using merely their own sensors to determine and predict other robots actions. Finally there may be *no synchronization* mechanisms provided for the group members. Usually the third scenario implies that either each robot acts independently, performing different tasks and treating the others as additional obstacles, or that collective behavior emerge from strict rules defined for the individual behaviors.

**A modeling of the other robots** is usually interdependent with the previously mentioned issues. For example robots working in a heterogeneous team relay their performance on the information about the services that can be acquired from the other members. The efficiency of the robots synchronizing via environment depends on how precisely they can interpret and predict the other robots' actions. In general the internal robots models can be characterized as either *capabilities*, *dynamics* or *primitives* oriented. The capabilities models define a robot in terms of its possible applications. For example as the lists of a high level services (task capabilities) available for the other members or sets of PDA (see section 2.1.1) operations for the planning system. The dynamics models address possible behaviors of the other members. A good example here is a convoy of autonomous robots where information on how quickly the ancestor can stop is safety critical. Dynamics are

usually represented as additional parameters or constraints in control equations. Primitives can be seen as a low level capabilities models. They allow one robot to make use of the others robots' control mechanisms. For example one robot can provide its proximity sensory data to the others and react to the requests to alter its positioning.

The existing publications on multirobotic do not come up with any new architecture for constructing the team members. On the contrary they focus on the interaction related problems defining collective capabilities of an individual as extensions to the most successful single robot architectures. Unfortunately particular changes that need to be applied are rarely described explicitly. One schema for such extensions, dedicated to the heterogeneous cooperative systems was described briefly in [96] by Simmons (see Figure 10). It was defined as a natural extension to the three layer architecture (section 2.1.3).

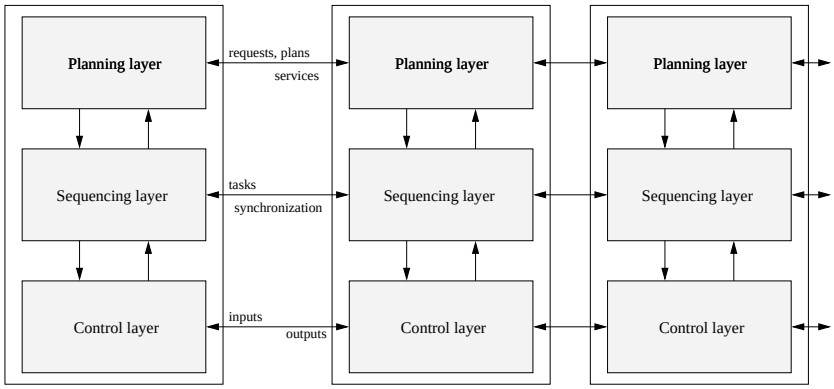


Figure 10: Team communication extensions for the three layer architectures

On the planning layer, the communicating robots can agree on shared plans, exchange offers or requests for particular services. The messages sent on the sequencing layer can be used for synchronizing the robots tasks and activities. The lowest (control) layer communication allows team members to form distributed feedback loops. For example one robot equipped with sophisticated vision system can directly provide its measurements and analyzes about external environment to the other participant. The described schema outlines another challenge associated with multirobotics architectures. The communication within a team can be hold over many levels of abstraction referring to for example: structure and contents of the plans, task exchange, activity synchronization, negotiation of individual rewards, sensor outputs sharing. Many of these subjects require guaranties on the link throughoutput and message delivery as well as constraints on the worst delivery

time. A protocol addressing all these issues is challenge. There are several publications related to this problem in context of multirobotic domain. In [35] Durfee and Montgomery present a hierarchical protocol for efficient exchange of information on different levels of abstraction. Davis and Smith [29] describe the *Contract-Net* protocol for automatic negotiation. However the performance aspects of both mentioned solutions are not addressed, which makes them less suited for real-time applications. A set of coordination protocols addressing real-time isues where provided in [25] together with a *Shared Activity Coordination* (SHAC) framework for distributed negotiations.

Several different approaches for providing an overview of the multirobotic domain appeared in the literature during the past few years. In [75] Matarić provides an overview of multirobotic domain in relation with its historical roots and inspirations (e.g. Distributed Artificial Intelligence). Cao et al. [22] contains a rich bibliography referenced to both the most popular task domains and the applied architectural solutions. Dudek et al. build in [34] a taxonomy of multirobotic solutions using groups size, communication topology and range etc. for characteristic. Parker describes in [88] "state of art" considering briefly both origins, architectures and typical applications. In the following sections we present various existing architectures taking different collective models for the reference and outlining changes needed to extend selected single robotic architecture with a team work capabilities.

### 2.2.1 Collaborative model

Collaborating robots form a logically associated group, sharing a common plan and coordinating its execution. They synchronize their actions on the task level. I.e. if $P$,$Q$ are two actions planned as sequential, then no robot starts the task $Q$ until $P$ is reported finished.

In the generic version both planning and the execution control of the job is centralized. The planner prepares a detailed plan as directed graph of required tasks (e.g. defined as specific robot behaviors). The tasks are then sequentially assigned to the different robots according to the model of their capabilities. Whenever the robot accomplishes its task, it reports to the planner and the next consecutive tasks can be delegated. That scenario resembles a deliberative single robotic architecture with the slight difference that the plan does not need to be serialized for execution and unrelated tasks can be run in parallel by a teamed robots. The centralized approach with a detailed task level planner should be treated rather as a reference model. The complexity of the planning problem is (similarly to deliberative) generally exponential and a large number of tasks make that solution in most cases intractable.

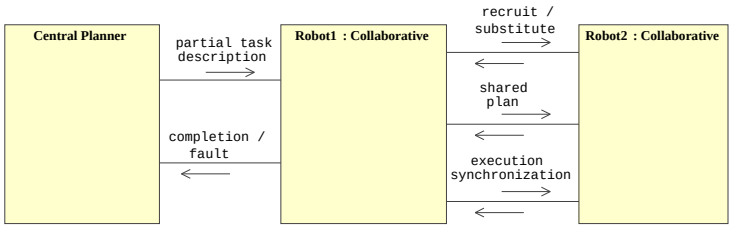


Figure 11: Communication in a collaborative model.

In practice, the centralized planner operates on a higher level of abstraction, using merely a partial description of the constituting tasks. For example the tasks can be stated declaratively with goals to be achieved (see Figure 11. Such a description is delegated to the robotic groups for further refinement. The teamed robots formulate individual plans and merge them into a single group-wide plan. The tasks are then divided between the members for execution. The plan merge and the task allocation can be accomplished either centrally by some "leader" robot or in a decentralized manner using a specialized negotiation protocol. Described schema improves the generic approach in several ways:

- Hierarchical planning allows to cope with the complexity of the problem solving and increases responsiveness to local environment changes.
- Error recovery can be implemented on three different levels increasing fault tolerance of the system. The robot can handle errors individually e.g. by executing alternative behaviors. It can secede the task to the other member. Finally, the group may decide to report an inherent fault to the central planner.
- Groups can be formed dynamically and change over time according to the assignment requirements. As the robots capabilities are not modeled and strictly attached to the task by the central planner, the group can optimize resource utilization for example by recruiting new members or releasing unused ones.

Jennings and Kirkwood-Watts in [56] describe an approach to constructing collaborative teams called *Method of Dynamic Teams*. They focus primarily on the organization and synchronization principles for building flexible multirobotic solutions. According to them : *"A dynamic team of agents is a temporary and fluid team whose association properties are allowed to vary over time. That is, teams dynamically and automatically grow and shrink, and members may be substituted. Also, an agent may be a member of more*

*than one team at a time."* The proposed method is supported by a specialized language *Scheme* that introduces primitives dedicated to multirobotic collaboration such as: `recruit` used for getting new member into a team, `with-team` selecting for example the best suited agent for task execution or `substitute-if-possible` providing error recovery capabilities on a group level. *Teambotica* [112] is another example of an architecture for collaboration inspired by distributed patrol-tracking missions. It defines mechanisms for collaboration based on the concept of *Shared Plans* [49]. Shared Plans is a formalized theory of a group planning expressed in a modal logic. It defines plans in terms of "intentions" to achieve goals and "beliefs" about the context (e.g. other robots commitments) rather than as a strict relation between planned tasks and required actions. That allows the system to refine plans iteratively until desired goals are met. The consensus for the plan structure and the task assignment is reached in several steps of negotiations interleaved with planning. A similar approach is presented in [2] by Alami et. al. They also describe incremental plan negotiations adapting the *Contract-Net* protocol for communication. Moreover some new optimization mechanisms like opportunistic action reassignment (which allows one robot to execute the action planned for another one) or detection and suppression of redundancy (that allow to reason on and to remove the unrelated actions resulting in the same state) are discussed.

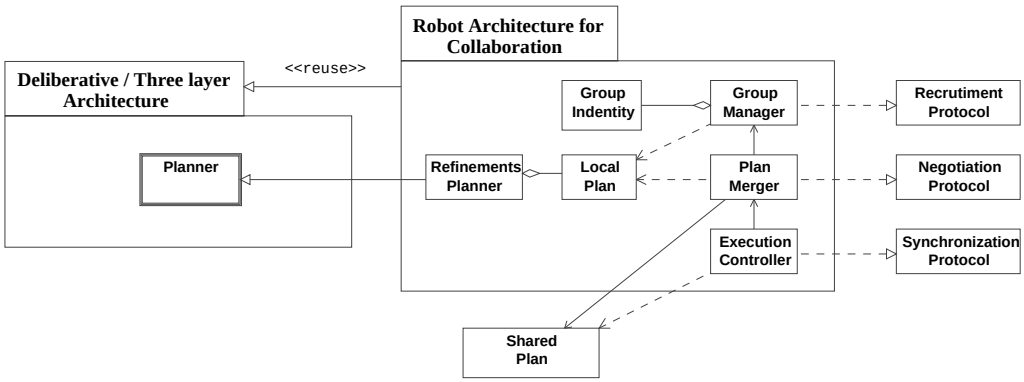


Figure 12: Architectural extentions in a collaborative model (class diagram).

The collaboration capabilities are usually implemented as extensions (see Figure 12) to either deliberative or three layer architectures. Indeed, the concept of plan refinement and sharing requires some reasoning capabilities typically implemented on the level of a deliberative planner. Also the execution control is expressed in terms of high level tasks that are: strictly ordered, explicitly assigned and contains precise constraints on completion.

It makes collaboration less suited for reactive architectures or motivation networks in which the behaviors are context dependent rather than explicitly stated. It is worth mentioning that the negotiations in the collaborative model are focused on optimizing the plan for overall job execution (threated as a scheduling problem) rather then searching for the best suited robot for a given task. It implies that the performance measurements on how well the robot can do the task are not crucial. The simple information whether the robot is capable of doing the task is in most of the cases sufficient.

### 2.2.2 Cooperative model

Cooperation can be considered as an extension to collaborative or competitive architectures. The main improvement is that cooperating robots can share a single task assignment and join their forces upon successful completion. It is achieved by communication and synchronization on a level of primitive behaviors or even on a level of a control data. The cooperation may be either *implicit* when the robot relies on the behaviors of the others, yet maintaining its own autonomous control or *explicit* when the robot actions are driven by direct requests from the other members. The implicit cooperation usually requires the robot to model the other members behavior. It is needed in order to identify their intentions and select appropriate reactions. Communication protocols are often used for synchronizing robot states and control modes. Explicit cooperation is particularly useful if the robots provide different, complementary skills. In that case cooperating robots can be treated as a single but physically distributed system with a well defined flow of control. Such an approach assumes that robots maintain a low level capabilities model (primitives model) of the other members and can interface to it, receiving low level data or enforcing particular actions.

Luis Chaimowicz et. al [24] present an architecture for tightly coupled multirobot systems with an implicit cooperation. They explain their ideas using the example of two robots carting the same object. The robots can work in several modes related to the primitive behaviors such as approaching to the object, lifting, and pushing or pulling it to a given location. The transitions between modes are guarded by a synchronization protocol providing information on the other robot's state. Moreover, during the transportation phase the robots dynamically can take one of the roles of either the leader or the follower. The leader selects the pulling behavior and takes initiative in altering the box position. The follower selects the push mode and focuses on the reaction to the box position changes in order to maintain stability. Reid Simmons et. al [97, 98] present an example of a heterogeneous robotic team cooperating explicitly on a large-scale assembly tasks. The described team

comprises three autonomous robots. A "robotic crane" capable of lifting and carrying heavy objects, a "mobile manipulator" capable of precise positioning of the lifted things and a "rowing eye" providing high resolution vision. The assembly task is accomplished by explicit cooperation using direct communication where one robot can request a particular behavior or a primitive action from the other. For example, the mobile manipulator can force the crane to move the object if it is too far away to be assembled correctly, or request the rowing eye to move the camera in order to gain a better view of the environment.

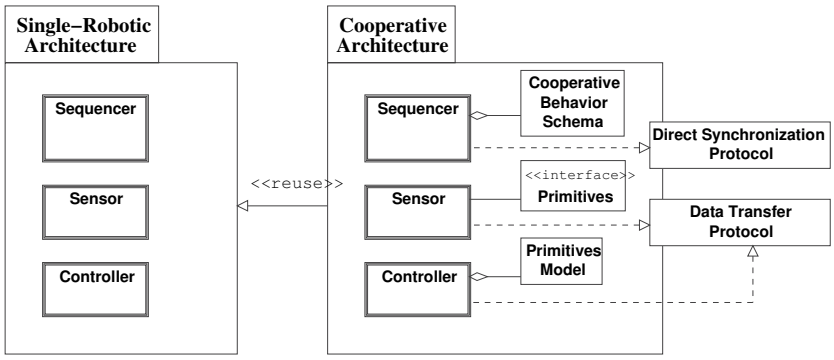


Figure 13: Architectural extensions in a cooperative model

Cooperation can be implemented as a single robotic extension on the level of primitive behaviors (e.g. sequencer lever). In that case the robot should select the actions that reflect the other robots' behaviors. A typical extension is then explicit model of the other robots' behaviors supporting interpretation of "what are the other robots doing?" and selection of a special, cooperation dedicated behavior accordingly. It is possible that the robots communicate their intentions purely indirectly "via environment". However, such a scenario appears often too complicated and error prone as the information gained by observation may be insufficient to interpret the others actions properly. In order to address that problem, direct synchronization protocols are provided and used for exchanging explicit statements about the robots' states and intentions (planned behaviors). Another possibility for cooperation is to make the lowest level control primitives public. In this case each robot can receive the sensory inputs of the other robot or influence others control modes. Such a solution requires explicit interfaces to the self provided services, information about services available from the others and efficient protocols (usually also high-bandwidth connections) for communication.

### 2.2.3 Competitive model

In the competitive model, team members are aware of the tasks constituting the problem solution and can compete with the other robots for task assignments. Although the robots are assumed to be self interested, i.e. they try to maximize their profits which usually affects negatively the other robots' profit, the model does not exclude collaboration or cooperation if such could appear more beneficial for the participants. An inherent feature of the competitive models is a separation of a centralized, high level planner from execution mechanisms. The tasks execution is distributed between the team members with no single point of control. The planner usually publishes tasks to the team members via some broadcast mechanism. It can influence the execution only by decisions on timings for new broadcasts (which are based on notifications of already accomplished tasks). The robot-task assignment problem is resolved by some election or negotiation protocol among the team members.

The described schema has several important consequences influencing its applicability.

- Planning procedures can be simplified and expressed declaratively. As the robot assignment is resolved non-deterministically from the planner perspective, it makes no sense to model team members capabilities within the problem solving procedures. It can also be perceived as a drawback in a highly heterogeneous team, where utilization of such information would lead to a significant plan improvement.

- The robotic team is flexible. It can grow, shrink or substitute its participants without affect on the plan execution.

- There is an implicit assumption that robotic forces are sufficient to accomplish all the specified tasks and no re-planning is needed due to the task ambiguity or the lack of forces.

- Execution efficiency and resource utilization are considered issue. There is no explicit way to assign task to the idle robots or optimize execution times.

- Another important aspect often discussed in context of competitive models is fault tolerance. Whenever the robot cannot accomplish or is ineffective in performing the assigned task (either due to system malfunctions or task complexity) an execution of the entire plan can be negatively affected and potentially stuck.

The strategies for task assignment vary from simple, based on numerical utilities to complex multidimensional negotiation protocols. In [90] Lynne E. Parker presented a simple distributed strategy for resolving assignments. In principle each robot stores available tasks in a vector and broadcasts its "motivation" toward performing each of them. The motivation for each task is a mapping from the robot capabilities, internal state and historical data into a subset of real numbers. It is assumed that the robot with the highest motivation for the task is the one assigned to it. As the motivation value varies over the time, e.g. it can decrease for the robot that is unsuccessful in its task achievement attempts, or increase for the other robots as expression of their "impatience". The method guarantees that a robot incapable of achieving an undertaken task will eventually be dismissed by another team member. It solves the problem of malfunctioning robots and inappropriate assignments. However, in case of ambiguous tasks that can not be accomplished (e.g. due to the changes in external environment) the assignment will circulate among the members unclassified as a 'livelock' in the plan execution. The market approach is another strategy for a task assignment presented by Dias and Stentz [101, 31, 32]. They introduce 'economy' in context of robotic teams as "a population of robotic agents producing a global output". According to their approach all the possible outcomes of some plan can be mapped into revenue values by some function $F_0$. The function is intended to evaluate predicted results within a range of tolerable deviations from desired goal, e.g. the size of visited area in scouting or patrolling mission, percentage of collected objects in foraging mission etc.. Similarly all the possible schema of performing the job can be mapped onto the cost values by some function $F_1$. The cost values reflect the effort needed to accomplish the task measured by for example total power consumed by the robots or time to finish. The idea is to find a plan (together with assignments) $P$ that maximizes the profit defined as $F_0(P) - F_1(P)$.

A mechanism for distributing task assignments assumes that each robot maintains its own "budget" comprising of revenues $f_0(T)$ acquired for performing the task $T$ and cost functions $f_1(T)$ providing an estimation of the expected execution effort. The cost function for a given task can be constructed and modified basing on previous measurements for similar task. The robots can compete for the assignments bidding different revenue values for the same task. Several auction protocols (see below) can be used to select the best contractor. Moreover, each robot can offer part of its own assignment to the other participants, whenever it may increase its profit. For example suppose that a robot assigned to some task is getting a revenue equal to $r$ and can accomplish the task within the cost equal to $x$. Its profit is thus $r - x$. If there exists another robot capable of doing the same task with cost

$y$ where $y < x$, it can be more profitable to sell the assignment for some price $z \in < y, x >$ and gain the profit $r - z$. The robot can negotiate with each other over the $z$ value in order to maximize its profit for such a deal. A similar schema can be used for organizing the robots into cooperating groups that shares the same single task.

Potentially, market economy mechanisms can lead to the situation in which the robot (or group of robots) most capable of doing the task is getting the assignment, i.e. the negotiation process ends up at Pareto optimum[4]. It provides a fully distributed architecture with self organizing capabilities. It also illustrates well the competitive model and gives a rich area for experimenting with various negotiation strategies. However, it has not yet been implemented and proven in general on realistic examples and many approach issues like fault tolerance mechanisms or general execution efficiency require further investigation. Also principles for selection of the most efficient negotiation subject and schema remain an open question.

The negotiation problem can be considered as a distributed search problem in a multidimensional space, where several agents are making an attempt to find a common space i.e. mutually acceptable agreement. There is a large number of possible options and approaches in relation to the negotiation problem. For example it can be organized in form of an auction (e.g. English, Dutch, First-Price Sealed) or as proposal - counterproposal dialog. The participants can use concession (weakening their demands) or adaptation (reformulating them). They can be rational (acting to maximize their own gain), cooperative (willing to accommodate others demands if they do not contradict theirs) etc. In general negotiation is a phenomena discussed in many different areas, e.g. Social Psychology or Game Theory. Problems of automated negotiations have also been discussed in a number of publications. Mayer et al. [82] build a qualitive framework for finding and agreements over demands expressed as logical models. Jennings et al. [57] provide a rich overview on principles of automated negotiations extended with an amply references list.

The competitive behaviors can be implemented over the most of the single robotic architectures described in the previous chapters (see Figure 14). For example Parkers Alliance [90] uses motivation networks as an architecture for individuals, while Simmons [96] extends the three layer architecture applying market economy for a tasks assignment. Probably, the most problematic (complex) extension would be the one based on reactive architectures. It is due to the hardwired robot tasks and event driven behavior.

[4]The situation in which none of the robots robot can increase its own profit without making other robots profit smaller.

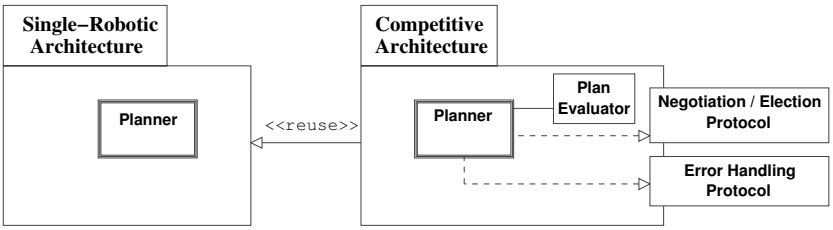


Figure 14: Architectural extensions in a competitive model

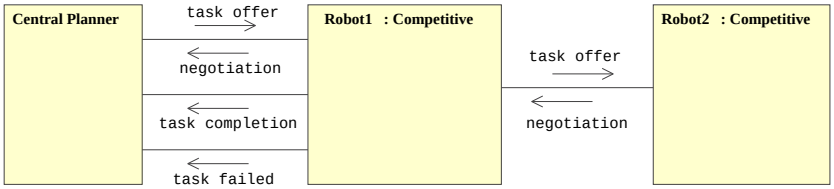


Figure 15: Communication in a competitive model

In general the teams of competing robots (see Figure 15) differ from a single robotic solutions mostly in the way they get the task assignments and handle the errors. Instead of performing the task given apriori by some planner (operator, task generator), they need to satisfy the task requirements beforehand and win the competition with the other robots. The architectural extensions contain a mechanism for computing a measurable estimate on how well the robot can perform a given task. They should comprise a negotiation or election protocol for getting the task assignments and/or trading them to the other members. The failures in task achievement are also handled differently in competitive models. Instead of reporting the error immediately to the operator, the robot should be able to retreat from the task offering the assignment to the other (perhaps more competent) member. Another protocol is thus needed for implementing distributed error recovery.

### 2.2.4 Emergent model

The emergent model was inspired by the observations on the creatures like ants or termites. They are capable of forming complex societies and raising sophisticated constructions, despite of being primitive insects, considered individually. The model is also closely related to the research in distributed algorithms, grid computing, e.t.c.. In the previously presented architectures the problem solving was associated with a formulation of the plan represented by strictly ordered set of steps needed for a goal achievement. In emergent the main focus is put on finding such a set of simple social rules

(constraints) and behaviors for the individuals that emerging group interactions will deterministically lead to the expected results. The emergent behaviors are built over a physical phenomenon that still lacks a constructive mathematical model. The principle idea is that each robot operates independently, acting on information, locally available through its sensors. The goal achievement capabilities are thus appearing from the local interactions. However, the goals are never explicitly stated. They are rather an inherent feature of the behavioral rules - an emergent invariant.

Lynn Parker presents a setup of his Alliance [88] architecture (described in the previous section) in which motivation values broadcast by the robots do not inflict the other robots decisions. In that case selections of the behaviors are merely dependent on the internal state of the individual and the stimuli received from coordinator. The setup was successfully tested for the group of three robots collecting scattered objects. Maja Matarič [73, 76] describes teams of reactive robots implementing a set of specially selected behaviors (called basic behaviors) dedicated to the particular tasks. The design was tested on a group of 20 robots for the tasks like aggregation, following or homing (heading to a given location). Rajeev Alur et. al [4] describe basic controller rules for the leader-following tasks. John Sweeney [106] discusses control primitives for the group of mobile robots maneuvering to remain in the line of sight (simulation of the group exploration with a limited communication range).

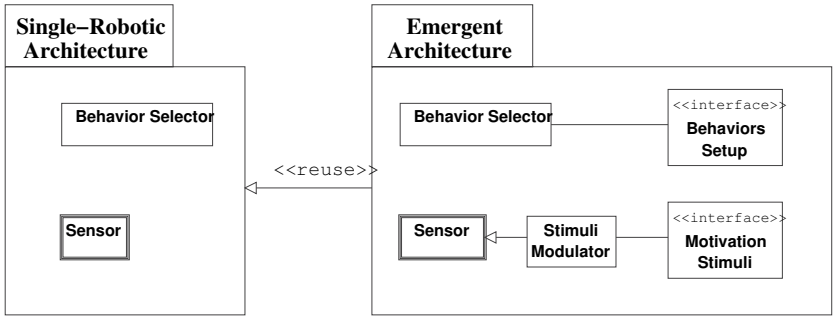


Figure 16: Architectural extensions in an emergent model

The emergent model is most naturally implemented as an extension to either the programmable reactive architectures or motivation networks (see Figures 16 and 17).

The central coordinator controls the job providing new primitive behaviors and control laws for the individuals or modulating stimuli that increase/decrease individual motivations for particular behavior. The changes to the single robotic architecture, require explicit interface for download-

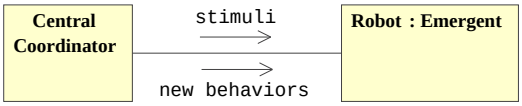


Figure 17: Communication in an emergent model

ing new behaviors or "acting as additional sensor"interface for introducing coordinator stimuli into a motivation network.

# 3 Machine Learning

Machine learning is a process in which the robot behaviors change over a time converging toward some optimal configuration. For example, a robot can modify its reaction delays in order to adapt to the environment conditions, devise different behavior selection rules to minimize the number of failures in goal achievement, come up with new strategies for negotiation to acquire more suitable assignments. In theory all the desired optimizations could be elaborated beforehand as an engineering tasks during the design and construction. However, in practice such an approach is usually not feasible. Human elaborates solution based on its own senses and perception of the world. The notion of optimality devised in that way may be simply not adequate for the robot equipped with sensor and actuator mechanisms that are inherently different from the human ones. Also theoretical models of the environment (together with formal reasoning mechanisms) provide a limited help, as the optimization is usually associated with physical robot properties that are by default abstracted out of the model. Another argument against the fixed behavioral settings is that the environment conditions tend to change (along with the optimality notion) enforcing the robot to adapt respectively. For example the mobile robot should be able to adjust the driving speed according to how slippery the terrain is.

The learning and adaptivity mechanisms are being intensively analyzed for the applicability to the various aspects of a robotic design. Recently the most popular topics include:

- *Optimizing primitive behaviors* that covers adapting the control rules to changes in environment conditions or the adaptive sensors fusion algorithms. Franklin [41] shows an example of refining motor control of robot for nonlinear tasks. That sort of optimizations find many commercial applications not only in the autonomous robotic area. Reliable movement detectors or steering support systems are only some of such examples.

- *Learning new primitive behaviors* is considered as an interesting option for the systems in which the environment-robot interactions are poorly understood and only basic expectations on the robot behavior are provided. Noisy sensors and imprecise actuators can hamper the efforts of modeling the robust robot controllers, yet such a controllers can be effectively devised in the training process. Papers presented by Harvey et al. [51] or Watson et al. [114] describe experiments of evolving controllers (encoded as neural networks) dedicated to obstacle avoidance and scouting behaviors.

- *Learning composite tasks* is associated with finding a sequence of the primitive behaviors leading to a predefined goal. It is considered as a crucial aspect of the robotic autonomy. For the robots working in complex and constantly changing environments implementation of all the possible scenarios beforehand is simply impossible. A number of publications [108, 20, 71, 99] (discussed in the following sections) address various experiments and strategies for that problem.

- *Optimizing composite behavior structure* is associated with an exploration of alternative paths in a task solving problem. It is usually accomplished by introducing random changes deviating from already found and proven solutions. For example learning robot can remove primitive behaviors from the behavior sequence and try the others instead.

- *Modifying behavior constraints* is somewhat similar to the problem of optimizing the primitive behaviors. According to the observations from experiments the robot can update the information on preconditions and results of the taken actions. For example Maes and Brooks [68] describe a method for effective selection of the most reliable behaviors. For each behavior they maintain constantly updated feedback data measuring number of times when certain condition were satisfied or violated after the behavior execution.

- *Learning cooperation and collaboration strategies* copes with a problem of finding optimal rules for a joint task execution. It also addresses the exchange of experiences (devised control programs, behavior sequences e.t.c.) in order to reduce overall learning time. Tan [109] presents results from both cooperation and experience sharing experiments. Makar et al. [69] provides another example of two robots collaborating on a trash collection task, where the division of responsibilities is learnt from experience.

In general learning can be defined as a problem of finding an optimal state space partitioning together with a mapping into a set of actions (control rules, behaviors, state transitions, e.t.c.). Intuitively each abstraction class represents a set of conditions for which the assigned action was found optimal (see Figure 18). The elements constituting the state space can be arbitrarily complex including sensory data, internal state of the robot, historical data, information communicated by the other robots, goals committed as user commands. One one hand it is tempting to contain more information in the states as it may lead to more fitted correlations and as a result a better performance. On the other hand the learning time is rapidly growing with the state space size, thus it is reasonable to keep them as small as possible. Finally most of the environments, where the autonomous robots are applied, are only partially observable (a part of important information is inherently hidden). It makes finding an appropriate level of abstraction a challenging task per se.

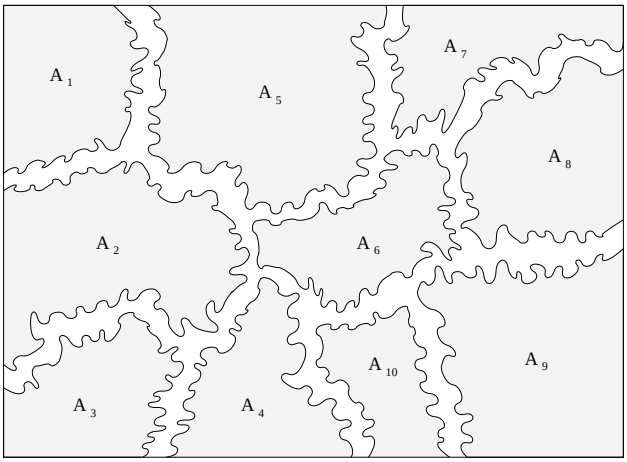


Figure 18: Learning in autonomous robots domain as a problem of building the partitioning and mapping of the state space. $A_i$ denotes optimal actions assigned to subsets of the space. The white areas represent adaptivity or error regions.

Although most of the learning methods discussed further in the text are common for all of AI domains, the applications in autonomous robotics are significantly different both on the assumptions on the learning goals and the interpretation of theoretical results. The learning problem in the domain of autonomous robots can be characterized by the three main elements:

**The Learning process** treated as a partitioning and mapping task is based on exhaustive exploration of the state space. Unlike in the other (espe-

cially software agents oriented) domains, the experiments with physical robots take a long time. The part of the state usually reflects partially uncontrollable environment variables. It affects the distribution and restricts the coverage of the experiments. As the number of the states explored during the learning process is severely limited, the theoretical convergence of the selected method is less important than the ability to find any solution within a limited time.

**Adaptivity** is associated with capabilities of reacting to the changes in these environment conditions that cannot be directly expressed (measured or interpreted) within a state space. The learning methods discussed in AI assume that a complete information on the current state is available and that the actions lead to deterministic results (i.e. changes in the state space). Indeed considering for example the problem of learning a chess game: position on board captures all the relevant information necessary for deciding on the next move. Each move changes the position deterministically. It is possible then to discuss learning process in which the number of nondeterministic choices is constantly decreasing potentially stabilizing in a limit on some optimal strategy. On the contrary, the autonomous robots neither operates on a complete state information nor provide deterministic results. More often it is possible to observe repeatable results that tend to fluctuate over a longer period of time (together with an environmental conditions like illumination, terrain, humidity, e.t.c.). The learning process is not intended in such a situations to reach a single stationary solution but rather to capture the correlations that appear permanent, yet sustaining ability to adapt to the changing ones.

**Equipment and modeling errors** are also common sources of limitations in the autonomous robots learning. Even if the results of some action are always deterministic or under some circumstances one action is always better from the others, the feedback information may be inconclusive due to the lack of precision in sensor measurements. The modeling errors appears from the general assumptions on the learning methods. Although the robot working in external an environment experiences continuous changes, its observations and reactions are usually discretized in some arbitrary way. For example the notion "primitive behavior" refers to a set of simple actions that lead to some result within some time. Moreover we tend to question the possibility of providing more strict and still applicable definitions. Another questionable assumption is that the learning mechanisms has a perfect synchroniza-

tion with the environment i.e. the results are observable immediately after the action is taken.

Most of the learning methods appearing in the AI literature can be characterized as *reinforcement learning* (RL) methods. The RL relies on *reward functions* providing a scalar assessment of the taken actions. It can be expressed as a Markov Decision Process (MDP) that models agent-environment interactions. At each time-step agent observes a state of environment $s$. Based on this information the agent selects and performs some action $a$. It receives a reward for the action results communicated through a real-valued *reinforcement signal* $r$. The goal of the agent is to choose the actions in such a way that it maximizes the total cumulative reinforcement value. An important feature of the MDP models is that they apply to environments that satisfy a *Markov property*. The Markov property tells that the model is "memoryless". It can be briefly formalized as: Suppose that the agent observes the states $s_0$,..., $s_n$, $s_{n+1}$. In each state $s_0$, ..., $s_n$ it takes respectively actions $a_0$, ..., $a_n$ and receives a rewards $r_1$, ..., $r_{n+1}$. Then the probability of reaching the state $s' = s_{n+1}$ and receiving a reward $r' = r_{n+1}$ satisfies (a classical Bayes equation for stochastic independence):

$$\begin{aligned} &P(s' = s_{n+1}, r' = r_{n+1} | a_n, s_n, r_n, ..., r_1, a_0, s_0) = \\ &\quad = P(s' = s_{n+1}, r' = r_{n+1} | a_n, s_n) P(s_n, r_n | a_{n-1}, s_{n-1}, r_{n-1}, ..., r_1, a_0, s_0) \end{aligned}$$

The decision on action taken by the agent in some state $s$ does not depend on the states the robot observed in the past and actions it took.

The RL methods can be roughly divided into two groups: *temporal differences* and *supervised learning*. The historical version of supervised learning (SL) is based on a paradigm of associative memory. The agent is presented with a pair of items. Later, when the first of the items is being shown, it is supposed to recall the second one. The paradigm is often applied for pattern classification or system identification problems. For the autonomous robots domain we shall provide alternative more suitable definition: The robot is supposed to learn an unknown function. It is presented with an argument for the function, provides its prediction on the function value and receives a feedback on how the prediction differs from the expected results. Then the robot updates its knowledge in order to provide more precise predictions in the future.

In many cases a feedback is a numerical reward value, and the robot task is to learn the prediction on rewards it receives for taking its actions in a given context. Having such a knowledge in each state it can take the action that guarantee maximal immediate reward. Assuming that the robot in state

$s_t$ executes action $a_i$ for which it expects the reward $F(s_t, a_i)$ while its actual value is equal to $r$, the learning procedure can be described as an update:

$$F'(s_t, a_i) \leftarrow (1 - \alpha)F(s_t, a_i) + \alpha r$$

where $\alpha$ $(0 < \alpha < 1)$ is a parameter determining a learning rate.

The supervised learning is a suitable method for adaptivity and learning primitive behaviors (see section 2.5 for example) where a greedy approach of selecting the action giving the maximal immediate reward is sufficient. However it largely ignores the sequential nature of the problems (where the future rewards may depend on a sequences of the taken actions) That makes this method less applicable for the tasks like learning complex behaviors.

In the temporal differences (TD) methods [104] the robot receives also a *reinforcement signal* immediately after some action is taken. Moreover the rewards for the following predictions are also included as a part of reward with some discount factor. The robot is thus rewarded for taking an action that will lead to the best opportunities in the future. The learning procedure can be in this case described as a recursive update:

$$F'(s_t, a_i) \leftarrow (1 - \alpha)F(s_t, a_i) + \alpha(r + \lambda F'(s_{t+1}, a_j))$$

where $s_{t+1}$ is a next state appearing after taking an action $a_i$ in the state $s_t$, $F'(s_{t+1}, a_j))$ is updated value of the following action and the $\lambda$ $(0 \leq \lambda \leq 1)$ is a discount parameter modeling the influence the future decisions have on a current one. The equation provides a foundation for a whole family of TD($\lambda$) methods depending on the value of $\lambda$. For the $\lambda$ equal to 0 the temporal differences method is equivalent to supervised learning. The TD(1) is a learning method in which the future decisions have equal impact on the preceding one. In the next section we shall discuss a well known model of reinforcement learning, called *Q-learning* that is closely related to the TD($\lambda$) classes where $(0 < \lambda < 1)$.

The learning methods are also discussed by Pattie Maes in [67] where classification based on action selection strategies, learning method and exploration strategies is provided. Also Sutton's book [105] discusses the basis of reinforcement learning methods illustrating them in a number of examples.

## 3.1 Q-learning

Q-learning is one of the most widely used learning methods in the autonomous robots domain. It was introduced in 1989 by Watkins [113]. The main strength of the method is its algorithmic simplicity combined with a proven convergence to optimal solutions.

A *policy* is a rule for selecting the next action in some time-step. For the MDPs the selection can be based merely on a perceived state and the policy can be formally defined as a mapping $\pi : S \rightarrow A$ where $S$ is a set of states and $A$ is a set of actions. For each policy $\pi$ and a state $s$ we define the *value of the state* ($V^{\pi}(s)$) as the total expected reward received.

Assume that for all states $s$, $s'$ ($s$,$s' \in S$) and all actions $a \in A$ the probability ($P(s'|s,a)$) of taking a transition from $s$ to state $s'$ after selecting the action $a$, is known. Moreover the immediate reward ($R(s'|s,a)$) for each transition is given. The value of state $s$ under policy $\pi$ is then defined by a recursive equation:

$$V^{\pi}(s) = \sum_{s' \in S} [P(s'|s,\pi(s))R(s'|s,\pi(s)) + \lambda V^{\pi}(s')]$$

For any MDP there exists at least one optimal stationary policy given by a solution to the Bellman equation:

$$V^{*}(s) = \max_{a \in A} \sum_{s' \in S} [P(s'|s,a)R(s'|s,a) + \lambda V^{*}(s')]$$

The equation determines the optimal value of state $s$ (thus defining the optimal policy for that state) as the maximum of the rewards received for executing each action, under the assumption that the value of the next state is optimal. The Bellman equation can be solved with standard dynamic programming methods. Such a learning strategy may compute the optimal policy off-line whenever a sufficient and reliable information on the environment model is available. In Q-learning gathering model data is part of the learning process. That makes it more applicable to the robotic domain, weakening the assumptions and addressing the problems of adaptivity and inconsistent reinforcement. The method is based on the following strategy:

For each state-action pair, the robot maintains a set of quality-values ($Q(s,a)$). After each execution of $a$ in state $s$ leading to perceived state $s'$ an update is performed:

$$Q(s,a) := (1-\alpha)Q(x,a) + \alpha[r + \lambda \max_{b \in A} Q(s',b)]$$

Where $\alpha$ ($0 < \alpha < 1$) is a learning rate, $r$ is the received reward, $\lambda$ ($0 < \lambda < 1$) a discount factor and $Q(s',b)$ the quality value of $(s',b)$ at the moment of update. One of the most important results in Q-learning says that:

Whenever consecutive learning rate values $\alpha_1$,$\alpha_2$,... ($0 < \alpha_i < 1$) converges to zero in such a way that $\sum_{i \in N} \alpha_i = \infty$ and $\sum_{i \in N} \alpha_i^2 < \infty$, and each pair $(s,a)$ is visited an infinite number of times, then the policy defined by $\pi(s) = \max_{a \in A} Q(s,a)$ is optimal.

### 3.1.1 Reward functions

One of the characteristic features of reinforcement learning methods is absence of an explicit environment model. Knowledge is encoded as reward functions and used to build a model during the learning process rather then explicitly expressed. The most natural way for constructing the reward functions is to provide a positive or negative feedback merely for these state-action pairs that end up achieving some goal or violating some constraints. Such functions are usually called *sparse functions* as they return a zero value in most cases (contrast to *dense functions*). Using sparse functions guarantees that the robot will learn only these solutions that are desired. However the sparsity raises some additional problems:

1. Most of the quality values comes from delayed rewards i.e. the value is based on a discounted rewards coming from some future actions.

2. The state exploration is mostly random, as no immediate reward is received.

3. The probability of executing a successful sequence of actions decreases exponentially with the length of the sequence.

As a result, if the required tasks are complex and the state space large enough, which is a common case, the learning process can run with a random exploration for a long time without showing any sign of convergence.

Matarić [74] addresses this problem by transforming sparse functions into dense ones. She introduces the notion of a "progress" providing an immediate reinforcement to the intermediate actions. In her example the robot learning how to get to a certain location is rewarded, whenever the distance between it and the goal decreases. Such an solution accelerates the learning and shows almost immediate convergence. However it affects the notion of optimality promoting these actions for which an immediate reward is available. It is also worth mentioning that with multiple alternative goals, construction of a single consistent measure of progress may be not feasible.

Smart and Kaelbling [99] suggest another solution based on a combination of supervised and reinforcement learning. In their example the robot is manually controlled by a human over the first several executions of some task. During these runs the robot observes the context and actions recording received rewards. That allows it to bootstrap the environment model and provides a direction for the further automatic learning.

Yet another approach reduces the number of states and actions that are locally used in a learning process. It is uses partitioning and building a

hierarchical structure over the state space. That kind of solutions will be discussed more widely in Section 2.4.3.

### 3.1.2 Policies

As explained at the beginning of this chapter, the policy is a strategy for selecting the next action. If the model (given by MDP) is known in advance, we can compute the optimal policy as a deterministic function from the state set to the set of actions. However, when the environment model is dynamically built, a certain strategy for determining the robot action is needed. In principle the robot can either *explore* the state space selecting untested actions and learn new solutions or *exploit* the best known patterns gaining certainty on their effects. The strategies for deciding on the next action in presence of uncertainty are known as *learning policies*. A learning policy should ensure sufficient space exploration, yet converging to optimality as model knowledge is gathered.

Many popular learning policies are based on an *exploration factor* - the value that defines the probability of selecting "not the highest ranked" action. The factor is being constantly decreased as the experiment time passes. Martinson [71] uses a simple formula $P_n = \gamma^n$ where $P$ is the probability of random exploration, $n$ number of the experiment time-step, and $0 < \gamma < 1$ the initial exploration rate.

Another learning policy [53, 109] is based on a Boltzmann distribution. The robot explores the state space taking action $a$ in state $s$ with probability

$$P(s,a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{b \in A} e^{\frac{Q(s,b)}{T}}}$$

The parameter $T$ ($T > 0$) modulates the frequency of deviating form the highest valued solutions. As the learning time passes the $T$ value is decreased to zero and in a limit the learning policy becomes equivalent to the optimal policy (or the set of optimal policies) given be the Bellman equation.

Policies based on always selecting the most rarely tried action, until the sufficient experience is gathered, are also effective [74]. They allow fast and evenly distributed state space exploration. They are mostly dedicated to robots operating in highly stable environments, where the action results are highly repeatable and external conditions known.

### 3.1.3 Hierarchical solutions

Hierarchical solutions gain increased attention as basic methods for improving efficiency and providing structure to the learning process. As discussed

earlier the "time to learn" grows rapidly with the number of states and actions involved in the process. Moreover, when longer action sequences are required for success there is a little the chance that the robot would eventually execute one of them. Hierarchical methods address these issues providing abstraction and generalization mechanisms. The concept is based on decomposition of a single, flat problem-learning model into a number of *nodes* - small, specialized models intended to learn a particular primitive behavior. Nodes can be further grouped forming new nodes, capable of learning more complex behaviors. The benefits usually discussed in context of the hierarchical methods include:

1. *Action set reduction.* The local learning mechanism may explore merely a subset of actions available for the robot. For example a node responsible for driving the robot to a predefined position does not need to explore actions associated with manipulating a robotic arm.

2. *State abstraction.* A flat learning model uses an unified state representation (encoded by the same set of variables) over the entire space. Decomposition allows to abstract irrelevant variables from a particular learning problem. For example a robot learning obstacle avoidance can operate on states containing a map delivered by proximity sensors, relative direction and speed variables, etc.. It may safely ignore global orientation and positioning. Learning how to grip an object can make use of the state of a robotic arm, relative distance to, and shape of a target, abstracting, e.g. a data from wide range sonars.

3. *Decomposition of sequential tasks.* Some tasks can be naturally divided into a number of consecutive largely independent steps. A robot used in assembly tasks needs to: 1) get nearby a target, 2) grip it, 3) carry to the destination and 4) assemble it. As the conditions determining start and termination of each activity are easily definable; it is much more effective to see them as separate tasks.

4. *Reusability.* Each node encapsulates acquired experience and learning capabilities dedicated to a specific behavior. It can be treated as a basic block of an adaptive system providing basic modularity features. The concept of developing new robots building from a set of already taught nodes is one of the more recent.

5. *Multirobotic solutions.* Assigning a task to the most capable robot can be seen as a selection problem. If the teams are dynamic or the robots performance varies over time (due to individual learning capabilities)

then adaptive selection mechanisms are needed. In such cases a multi-robotic solution can be considered a hierarchical learning system.

6. *Multiple goals learning.* Non-hierarchical learning operates on a single state space using a monolithic reward function. In many cases defining a single function for a multiple goals may be not feasible. For example it is difficult to represent both obstacle avoidance and a box pushing/pulling behavior by a single reward function as they both refer to a physical contact although assigning opposite reward values. Hierarchical methods allows to cope with such a problems, dividing them into separate subproblems and learning how to switch them contextually on a higher level of a hierarchy.

Similarly to the classical reinforcement learning, hierarchical methods cope with the problem of contextual selection (or prediction). However higher levels of hierarchies operate rather on sequences of actions than state-action pairs. This kind of learning is usually classified as semi-Markov Decision Process (for example as a Partially Observable MDP or a Hidden MDP).

In most of the cases the hierarchy structure is manually designed and programmed into a number of subroutines. If the solved task is complex it is difficult to decide whether the acquired hierarchical policy is globally optimal (i.e. the flattened version of the solution is optimal according to the definition given for MDP in section 2.4. To address this problem, hierarchical learning methods uses a notion of *recursively optimal policy*. The hierarchical policy is said to be recursively optimal, if for each node its descendants are recursively optimal, or if the node has no descendants, it is optimal in the MDP sense (see sec. 2.4). Recursive optimality defines a certain criteria for a hierarchical learning convergence. However it assures only a sort of local optimality with no guarantees on the quality of the overall solution. A more efficient notion of *hierarchically optimal policy* was provided by Parr and Russell in [91]. They define a measure for policies assuming some fixed hierarchy (i.e. restricting the number of possible state transitions). Given a hierarchy $H$, and a general MDP model $M$ for a learning problem, the policy $\pi$ is hierarchically optimal if it is optimal for $M$ restricted to selections from $H$.

In general the hierarchical methods can be divided into two groups: *homogeneous* and *heterogeneous*. The homogeneous exploit the same learning principle over all levels of hierarchy, while heterogeneous vary learning mechanisms according to the application. A simple example of a heterogeneous method is a variation of the adaptive three layer architecture. The sequencer operates on primitive behaviors, each encoded as an adaptive Q-learning controller. If for some behaviors a number of alternative controllers is available,

it can prioritize selection on the basis of success/failure feedback. The robot uses Q-learning mechanisms on the control layer and a priority queue for the sequencing layer. A more sophisticated method was presented by Hoff and Beckey [52]. They discuss "coordination learning" which is de facto a problem of behavior selection. In their example, each behavior is encoded as a neural network forming a separate reinforcement learning model. Their action selection is based on adaptation of behavioral inhibition (i.e. a stimuli built as a response to feedback from individual behavior performance). The mechanism is inspired by the neurobiological phenomena of *long-term potentation* and its corollary *short-term potentation*.

Most of homogeneous methods adapt standard Q-learning mechanisms to the hierarchical version learning [33, 69, 53] or define hierarchical structures for classical MDP learning process [111]. The main problem addressed in these publications is: "how to update quality values on a higher level of hierarchy in order to provide convergence of some kind". Humphrys [53] presents a method *W-learning* method. In W-learning action selection problem is modeled as a competition between "agents" - basic Q-learning nodes. Each agent observes state of the environment and suggests an action it wants to be executed. The selection on the higher level nodes is then based on the values provided by the descendants. In principle each agent could submit its best quality value in a given state. However such a solution would imply normalization of reward function over entire set of agents, that are assumed to be largely independent. Instead, W-learning suggests using a difference between predicted reward (Q-value) and reward acquired if some, other then suggested, action is taken. That difference, called *W-value*, is maintained and dynamically learnt by each agent. The higher nodes uses a policy of selecting the action that maximizes summary W-value. Generally each agent stores an array of W-values ($W(s)$), one for each distinguished state. Whenever the action different from suggested one is executed, the agent observes the resulting state $s'$, experience passively reward $r$ and performs an update:

$$W(s) := (1 - \alpha)W(s) + \alpha[Q(s, a) - (r + \lambda \max_{b \in A} Q(s', b))]$$

Where $a$ is an action suggested in state $s$. Humphrys proves that W-learning method converges to a stationary policy. However, as that form of competition mechanisms is not suitable for learning of action sequences (the policy is being determined on the lowest level of hierarchy) it is difficult to relate W-learning to any optimality criteria.

Dietterich [33] presents a method called *MaxQ*. In this approach each node can be activated by its parent and run for some time (executing actions or activating descendants) until it reaches a terminal state. than the node

returns a control to its parent together with a cumulated reward value. The MaxQ-learning is based on decomposition of standard Q-values into two components: the immediate reward for taking the action and the cumulated expected reward (called completion function) received for the following actions until termination. Such a separation allows to differentiate between improvement owed to the descendants learning and the one coming from finding more effective sequences of a higher level execution. The MaxQ is proven to be recursively optimal (under some additional assumptions). Makar et al. [69] show the application of MaxQ to the collaboration problem in multirobotic domain. In their example two robots learn how to distribute activities and synchronize actions for a trash collection task.

## 3.2 Evolutionary robotics

The *evolutionary algorithms* [83, 48] are a general exploration method for finding optimal values (and arguments) of a complex nonlinear function. They appeared as a generalization of *genetic algorithms* allowing the use of arbitrarily complex data structures as the function arguments[5]. The evolutionary search is based on assumption that the analyzed function is locally monotonic i.e. optimal values can be obtained from the "almost optimal" ones by some small step modifications on their arguments. The basic operations involved in evolutionary algorithms include *crossovers* and *mutations*.

Let $F : X \to Y$ be a function into some ordered set $Y$ for which we are looking for the argument $x \in X$ such that $F(x) = \max_{a \in X} F(a)$. Then the crossover is any function $C : X^n \to X$ taking some elements of $X$ as arguments and producing a new element of the space. The mutation is any nondeterministic operation $M : X \to X$ returning a new element (usually by introducing a random change to the one given as an argument). The part of the evolutionary programming art is to define crossover operations such that they provides extensive, potentially converging, local search (i.e. having a set $S$ of initial values from $X$ it is possible to find some local optimum $F(a)$ such that $a$ can be represented as a term over the values from $S$ and superpositions of $C$). The task of mutations is to provide differentiation of the arguments allowing the search process to escape from suboptimal solutions.

A typical evolutionary algorithm starts from a set of randomly selected elements (called a generation) from the arguments domain. In each step it produces a number of new elements applying crossover and mutation operations to the elements from the last generation. The new elements are then

[5]The genetic algorithms restricts the function domains to the fixed sequences of binary values.

assessed (using the $F$ function). The number of the best ones (sometimes together with the best elements from the previous generation) is selected to form a new, better fitted generation. Under several additional assumptions (see [83]) the process converges to the optimal values of the $F$ function.

Evolutionary algorithms is an example of a supervised learning where a new solution is produced based on evaluation of some reward function. The system tries to discover the optimal relation between the environment, controller and received reward by extensive search over the controllers space. They can be applied to the robotic domain for finding solutions to nontrivial design problems. Harvey et al. [51] applied the method for synthesizing controllers dedicated to obstacle avoidance and scouting behaviors. They were evolving the controllers represented by neural networks. Each network was encoded as a sequence of "genes" encapsulating nodes, activation thresholds and a relative connections to the other nodes.The single controller was run several times on a different experiments and evaluated for its fitness. The overall experiment involved running 50 generations, each generation containing 40 distinct controllers. It provided some optimistic results on the applicability and effectiveness of such methods.

# 4 Architectural Frameworks

A *framework* is a flexible, dedicated development environment adopting the best practices and supporting technologies and standards applied in a specific domain. As the most general framework for computer scientists is for example Turing Machine, Petri Net or Timed Automaton, it seems that specialization for a specific area and automatic or (semi-automatic) tool support is a crucial aspect of every framework[6] While some of the ways in which a framework could support development are domain specific, we desire to find certain criteria also for mobile autonomous robot framework. Moreover some concepts can be adopted from the best practices captured in frameworks used in other domains and interpreted in the context of autonomous mobile robots development. The following list contains enumeration of important properties that should hold for a framework for autonomous robots development:

## Properties of General Frameworks

- *support for architectural patterns* - a framework should provide some embedded support for one or more autonomous architectures.

[6]The author doubts that a box of nails can be called a framework for kennel building.

- *hardware abstraction* - as each autonomous mobile robot contains a number of sensors, motors and controllers, some general, preferably common convention on their interfaces and specification is needed.

- *development methodologies* - a framework should constrain instantiations in commonly used modeling and development methodologies and notations (UML, VHDL, etc.), support languages for expressing plans, describing state of environment (e.g. ADL, PDDL).

- *incremental development* - a key aspect of a framework is support for manufacture and reusability oriented assembly rather than design and implementation from a scratch. It should be operate with modular "higher-level" abstractions (components, packages, libraries, etc.).

- *component interoperability* - a transparent built in communication mechanisms for connectivity of architectural constituents should be provided.

## Robotic Frameworks

- *real-time execution environment* - efficiency of any robotic solution is dependent on delay in sensors-actuator loops and responsiveness of control programs in between. The robotic framework should explicitly support a real-time execution environment (operating system) and its analytical aspects like scheduling analysis and worst execution time calculation.

- *responsiveness granularity* - It seems that various levels of a robotic architecture require different timing granularity some examples are: real-time, synchronous communication on a control level to sporadic asynchronous communication between sequencing and deliberative layer or between two collaborating robots. Also the size and form of the messages differs from layer to layer. There is a need for different communication solutions (protocols, mechanisms) for each of the commonly used types of communication.

- *machine learning and adaptivity* - Many robotic solutions make use of some standard learning mechanism (neural networks, evolution algorithms, Q-learning) to modify internal control structures. The framework could support adaptive solutions, providing customizable learning mechanisms.

- *sensor fusion* - As the sensors measurements are often inconsistent due to the different parameters of applied sensors, there is a need for an arbiter maintaining consistent input data.

- *hierarchical actuator abstractions* - As many primitive behaviors operate on actions involving control of many actuators at a time (usually we are interested in having a robot moving forward not having one wheel moving) there is a need for higher level abstractions (e.g. motion system) providing unified manipulation methods at various levels of complexity.

There is an increasing number of both commercial and non-commercial frameworks supporting development of autonomous robots. The following sections exemplify a few of such systems. The selection is intended to display various concepts and views on making autonomous robot design a systematic task. It also shows that there is no community-wide agreement on what support is needed and what abstraction level is sufficient for a successful development. Other well known frameworks examples include: `Saphira` [60] or `DRS` developed at Michigan State University. Tool supported systems like `Onika` [102] dedicated to development of robotic arms, or `CDL` (Configuration Description Language) [65] for reactive robots design allow rapid graphical construction based on combining blocks which encapsulate dedicated control programs. There is also a group of languages (and associated tools) that support robotic construction in a more specialized way. Languages like `ALFA` [42] or `REX` [59] are capable of expressing reactive models that can be directly compiled into control programs. `Gapps` [58], `Supervenience` (APE)[100] are examples of declarative languages for expressing deliberative tasks and symbolic environment models that can be automatically compiled into planning programs. Biggs and MacDonald present an overview [8] on other robotic oriented languages. These could arguably be classified as restricted frameworks providing tool support for the particular step of a development. There are also many systems dedicated to hybrid modeling (classified in the text as generic frameworks ref. Section 3.4) such as `Omola` [77], `HSML` [110], `Modelica` [78], `SHIFT` [30], `VHDL-AMS` [54] implementations or a commercial `Saber`. The presented list is by no means complete. It is very likely that the number of offered solutions will grow rapidly together with a deeper understanding of developer needs.

## 4.1 BERRA

BERRA (BEhavior-based Robot Research Architecture) [64] is a project developed at the Royal Institute of Technology (Sweden). Its main objective is

to provide a research architecture for development of a service robot.

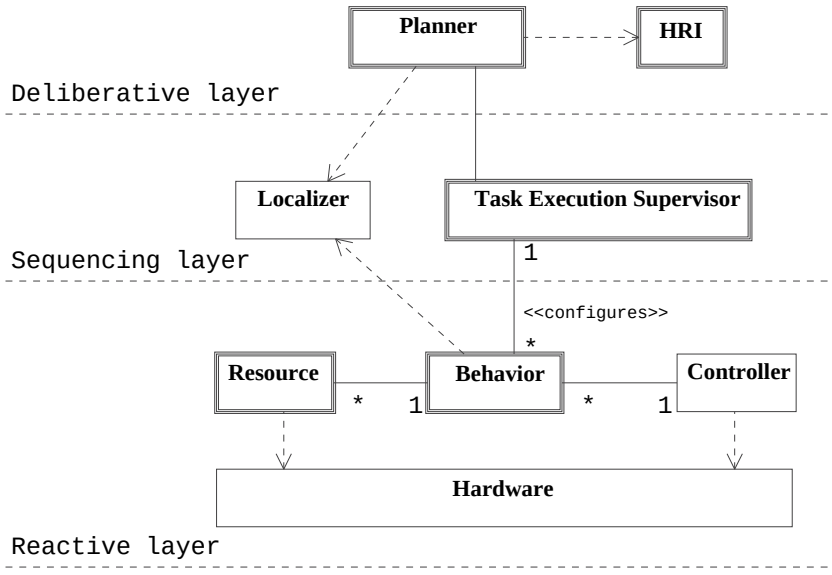


Figure 19: BERRA Framework

BERRA appears as an efficient implementation of a Three layer architecture (described in section 2.1.3). On a deliberative layer (see Figure 19) it comprises a `Planner` connected to a human-robot interface (HRI). The HRI provides speech and gesture recognition capabilities that are used for commanding the robot. The intermediate layer contains a sequencer mechanism (here called `Task Execution Supervisor` - TES) capable of assigning particular primitive behaviors to plan-steps received from the deliberative layer. It also provides a `Localizer` mechanism that interprets the state of the robot (positioning, successes, failures, etc.) in a form interpretable by a planner. The sensors has been abstracted in the form of `Resources` i.e. simple servers providing data to its clients - `Behaviors`. It is assumed that a number of independent behaviors can coexist in a control layer at a time. The Behavior in BERRA duffers from a concept of a primitive behavior presented in Section 2.1.3 in two ways: It is selected, configured and activated by the sequencer (TES) rather than being dynamically downloaded. It communicates with hardware control programs rather then replaces them. The *Controller* in BERRA forms a generic actuator abstraction that provides interfaces for action execution.

BERRA was developed on the top of a standard `LINUX OS`. It tries to exploit `Posix.1b`[7]. standard extensions (static priorities, unpreemptable FIFO

[7]Posix.1b standard (IEEE Std 1003.1b-1993, aka ISO/IEC 9945-1:1996) describes real-

queue for process scheduling, etc.) in order to provide real-time capabilities. The framework includes internal `Process Manager` (wrapped around a Linux scheduler) capable of managing execution of active objects in BERRA (behaviors, resources, etc.).

`ANSI C++` was chosen as an implementation language for the architecture. User programming to BERRA inherits from the basic classes for resources, behaviors or configurations. The active objects have assigned a unique IP address and communicate via a transparent, built in socket-based mechanism. Although, due to the selection of non real-time operating system the architecture may not scale well to more complex solutions, clearly defined developer tasks and competences, and a large number of supporting architectural mechanisms make BERRA an attractive solution.

## 4.2 LAAS

LAAS[8] (LAAS Architecture for Autonomous Systems) [1, 36] is another framework building over the concept of a three layer architecture. It supports systematic development of a single robotic systems. Unlike BERRA, LAAS integrates and makes heavy use of several independent development tools.

On the control layer LAAS incorporates `Generator of Modules` (GenoM) [39] used for defining basic control and processing loops. GenoM is a general tool to design and build real-time distributed systems. It aggregates system functionalities in *modules* - independent communicating servers and *codels* - elementary code chunks with an assigned thread of control encapsulating basic control algorithms. GenoM provides an unified execution control interface that allows to stop, start or re-parametrize the modules dynamically. The modules for a target system are automatically generated from some implementation independent model. Hardware and operating system abstraction is achieved by delivering a number of templates dedicated to a particular operating system and/or hardware architecture.

The sequencing layer in LAAS realizes two distinct tasks: 1) translates a higher level plan into a sequence of actions (requests to particular modules), 2) controls action execution.

The translation is implemented with a tool called `Propice` [55]. It maintains a database of facts about the environment state (localization, orientation, obstacles map etc.) that is dynamically retrieved from sensors. It also includes a static library of action sequences (procedures) leading to a predefined goal in a given context. Sequencing is achieved by a constant matching

time facilities for portable operating systems.

[8]Toulouse, France.

of upcoming plan steps and a recent state to available action sequences.

As the control layer contains of a number of independent modules, which are capable of executing redundant or contradictory actions, a mechanism for restricting or synchronizing primitive behaviors[9] is needed. LAAS uses a simple rule based system called `Kheops` [1] to generate the execution controller. In Kheops, the behavioral dependencies and interactions are expressed as sentences of a prepositional logic (rules). Generally Kheops uses monotonic deduction to translate the set of rules (execution model) into a finite state automaton (execution controller). The execution controller is generated offline during the development process.

The deliberative layer of LAAS uses `IxTeT` [55] - a planner based on refined logic formalism.

The LAAS framework has a simple development process. Users program different parts of the robotic architecture relying on provided tools. As some of the architectural components (modules and procedures) are abstract and mutually independent, the solutions remain flexible and can be reused in other applications. However, LAAS robots lack an uniform architecture description, as they contain of several parts expressed in heterogeneous languages. It makes fitting a conceptual design of the robot into the framework a laborious task.

## 4.3 OROCOS

The Open Robot Control systems (Orocos) project [18, 19] was initiated in 2001 as a joint enterprise of several universities (K.T.H. Stockholm, F.A.W Ulm, LAAS Toulouse K.U. Leuven). The goal for the project was to provide a general open software infrastructure (framework) for design and manufacture of software related artifacts dedicated to robot control systems. Such an infrastructure would both speed up the development process and form a common basis for knowledge exchange on the applied solutions and architectural patterns.

Orocos was designed as a two layered framework on top of a Linux based hard real-time operating system[10] (see Figure 20).

The Orocos core provides hardware abstractions in the form of C++ "interfaces" (abstract classes) defined for the most common robotic devices; e.g. actuated axises. The hierarchical actuator abstraction is also explicitly addressed by a set of interfaces for manipulation over commonly used actuator

[9] Similar to the "hardwired" suppress/inhibit mechanism in reactive architecture (sec. 2.1.2).

[10] There are three supported operating systems mentioned in the Orocos documentation: RT-Linux, RTAI and GNU/Linux

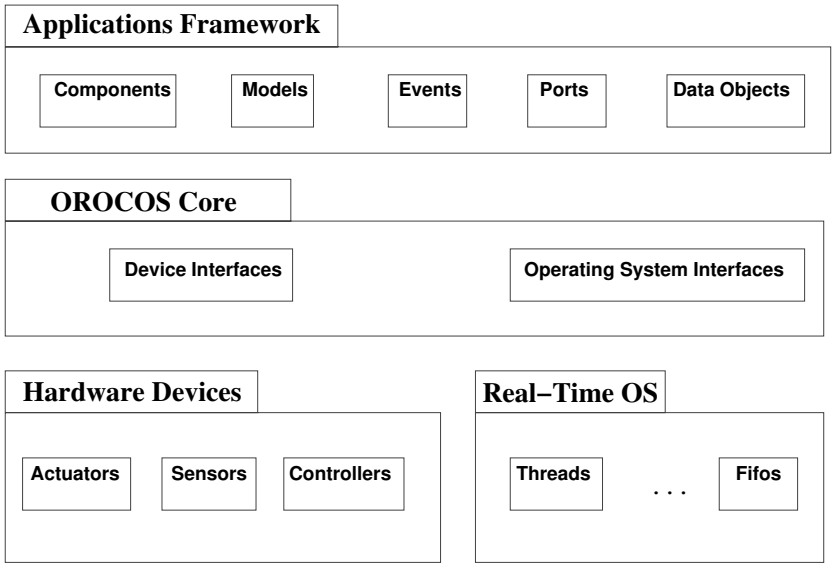


Figure 20: Orocos Framework

configurations, e.g. XY-table, serial arms, mobile manipulators. Another important feature implemented within the Orocos core is a real-time execution environment. It provides a facade over the real-time operating system scheduler defining three specific priority queues:

- `Zero Time` - The highest priority queue that is hard real-time and non preemptable.
- `Zero Latency` - The lower priority queue that is hard real-time but preemptable.
- `Completion Processor` - The lowest priority queue that is non real-time.

For the threads running on each of these queues the framework defines an abstract C++ class that needs to be inherited and implemented according to the interface. There are also other useful abstractions provided such as a "heartbeat" acting a common clock. Orocos does not provide any support for scheduling or worst case execution time analysis. Attempts to analyze the system on the level of operating system appear impossible due to the lack of information on the framework imposed overheads. The constraints and boundaries on the execution times of the threads remain merely decorative, which in the case of non preemptable threads is a quite dangerous concept.

The application framework layer introduces *a component* as a building block of a system communicating with other components via *ports* and *data*

*objects*. There are also other mechanisms like *events* and *fifos* (first-in first-out priority queues) provided for component interoperability. It is not clear whether the communication support is restricted to the framework and a single machine or can be used to communicate with other "third party" applications running either on the same or other network connected systems. There is no explicit support for the architectural patterns probably due to the generality and intended flexibility of the framework. Incremental development is partially supported by dividing the system into the components and imposing strict requirements on communication methods. However many years of experience with software engineering suggest that encapsulation based on published interfaces and informal semantics is not really effective and sufficient for component reuse and the incremental development[107]. Also mechanisms providing unrestricted access to OS specific system functions for the Orocos components invalidates portability assumption and appears as a serious drawback. Other framework concepts like development methodologies or sensor fusion are not supported.

## 4.4 Generic frameworks

The previous sections exemplified frameworks embedded in autonomous robots architectures. They were exploiting collected domain experiences, encapsulating the best practices and proven solutions into a higher level abstractions (e.g. actuator abstractions, primitive behaviors, layers). Another kind of frameworks commonly applied to the robotic development express robot architectures as a network of communicating and distributed *hybrid systems*. Generally, a hybrid system is a network of state machines where states change either continuously through evolutions or through discrete transiting. In such a modeling view, autonomous solutions are no different from any other embedded systems like telephones, light controllers, thermostats, printers or car cruise controllers. As embedded systems became pervasive, a number of formal theories [12] for expressing and combining various hybrid control models (e.g. Tavernini's, Back-Guckenheimer-Myers, Brockett's, Nerode-Kohn, Branicky) and concurrent interactions (e.g. Communicating Sequential Processes, Calculus of Communicating Systems, Continuous Time, Discrete-Events) have been developed. Together with use of frameworks that applies these theories, the domain knowledge on the architectural patterns is lost. However, the user benefits from capabilities to simulate execution and evolution or formally verify time properties of the design.

Typical examples of frameworks dedicated to a hybrid modeling include tools like `STATEFLOW`, `Ptolemy II` [63] or `Chronos` [3]. Although they all are equally expressive addressing the same class of systems, there are significant

differences in the general approach to the system modeling.

### 4.4.1 Syntax

Frameworks differ in the level of formalism offered in compositional algebra. For example, STATEFLOW model comprises of networks of *control blocks* that encapsulate discrete or continuous transfer functions. A block notion introduces an uniform, system-wide abstraction for composition. To complete the picture STATEFLOW supports hierarchical development i.e. aggregation of a block networks into a larger blocks. While STATEFLOW operates on uni-sort, blocks algebra, Ptolemy II combines a number of languages and computational models. One of the main Ptolemy II objectives is to provide a wide selection of modeling domains in order to make design a direct representation of physical system. From the overview [63] we learn that the models can be expressed by Communicating Sequential Processes (CSP), Discrete-Events (DE), Process Networks (PN), Synchronous Dataflows (SD), Finite State Machines (FSM), Continuous Time (CT), Discrete Time (DT), Synchronous/Reactive (SR) or Distributed Discrete Events (DDE). Ptolemy II allows intermingling of domains on different levels of the hierarchy. For example a thermostat model can be expressed as an FSM that alternates between an `on` and `off` state, while each of the states can be defined by a CT model with a thermometer and a heater acting in a feedback loop with the environment. The composition algebra in Ptolemy II is based on "bubble-and-arc" diagrams i.e. each domain model can be represented by nodes expressing *entities* connected by arcs describing *relations*. The exact semantics and syntactical restrictions to composition are inferred from a particular domain model.

In Chronos, the only subject for composition is called *agent*. An agent $A$ can be formally defined as triple $A = \langle TM, V, I \rangle$, where $TM$ is a set of *modes* i.e. building blocks for describing flow of control, $V$ a set of *variables* (either local or global), and $I$ - the set of *initial states*. The agents can communicate with each other through explicitly defined, shared variables. The agents can either be *composite* (constructed from other agents) or *atomic* (containing a single *dataflow* description).

### 4.4.2 Semantics

Different computational models and execution semantics semantics are provided. STATEFLOW copes with continuous time offering arbitrarily precise democratization. The execution is simulated by a number of consecutive steps. In each step the new system states (values of discrete/continuous func-

tions) are calculated according to selected progress interval. The Ptolemy II timed semantics are dependent on the domains (models of computation) applied in design. As a result Ptolemy II lacks a single consistent notion of time. However, it provides a formal notion of *progress* that enables interoperability and control flows between heterogeneous domain models. Chronos provides formal operational semantics for system execution. In general, each agent is running its modes resulting in either continuous or discrete changes. The time progress is expressed as a trace - sequence of alternating *continuous* and *discrete* steps. In continuous step, Chronos calculates the space of states that are reachable by applying only continuous changes constrained by an agent and environment models. Similarly in discrete steps state space reachable by merely discrete transitions is computed.

### 4.4.3 Validation

Third, the frameworks offer different system validation capabilities. Both STATEFLOW and Ptolemy II validates designed models by simulation. However STATEFLOW provides quantitive simulation, i.e. the system changes can be related to seconds, milliseconds etc. of the execution time. The number of different time modeling concepts applied in Ptolemy II makes such a simulation infeasible. It is rather qualitative providing capabilities of testing overall system properties such as deadlock freedom or safety. Strict formalization of time in Chronos allows to apply verification and model checking methods based on over-approximation of the model [3].

## 4.5 Summary

The described examples demonstrate that the framework characteristics provided at the beginning of this chapter is more a "wish-list" than a realistic description. Thus it is not obvious what should be classified as geniue robotic frameworks. Things get even more complicated, as the words like framework, architecture, language became *buzzwords* loosely used in technical literature. It reflects reality: Development tools support their own language known under the same name; standardized specification languages have several commercial implementations; notations are called languages; and architecture developers offer their design together with a number of dedicated packages, script generators, useful utilities.

In order to clarify the situation we shall provide a small road-map for what is known as a framework in the autonomous robots domain Figure 21. The idea in the structure of the picture is that the boxes from lower levels can be dependent (e.g. use, realize, refine, restrict, implement relations) on

arbitrary set of boxes from the upper levels. For example BERRA can be seen as an execution environment, implementing a three-layer architecture and providing layer specific language for reactive layer, namely for programming behaviors.

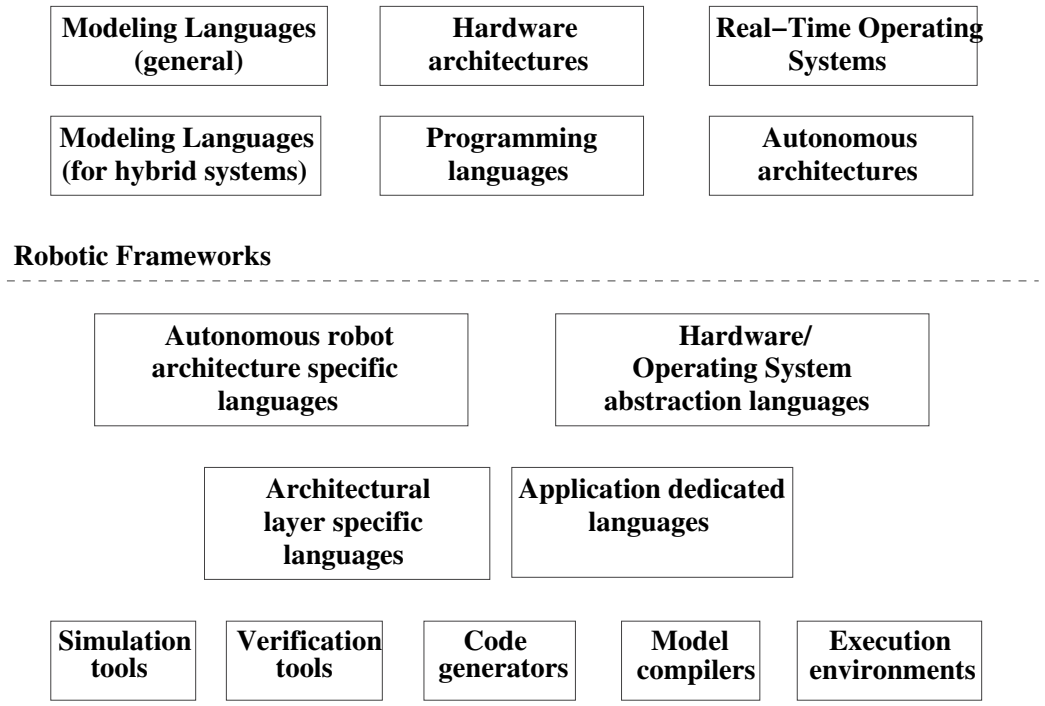


Figure 21: Kinds of systems that are often referred to as robotic frameworks.

What seems to be missing in all of the known examples is a general model of autonomous system (similar to the Three-tier model for Enterprise Information Systems [9]) which systematizes the development process and clarifies developer tasks.

Another commonly neglected problem is that autonomous solutions are part of a larger infrastructure incorporating external sources of information. Recent works rely basically on "embodiment", treating autonomous robots as black boxes extracting facts from sensors and reacting accordingly. That restricts demonstrated autonomous capabilities to skills like wall following or obstacle avoidance. It seems that for a qualitive jump in practical applications a broader view is needed. Taking the example from human world, a person moving to a new town, could learn it by individual random exploration. However it is more likely that he or she would buy a map. The autonomous robot needs to *import knowledge*.

*Context awareness* is illustrated by the following example: Dropping the

piece of paper in the open, one would hurry to grab it before the wind takes it. The same situation happening in the office room would not cause a similar rapid reaction, not only because one senses no wind on the skin. Our individual skills are thus not only based on perception but also on an ability to make use of the flexible context. It seems that support for integration of autonomous robot with external sources of information would help to demonstrate more complex applications.

Finally all of presented frameworks are dedicated to the single robot architecture, and do not provide any support for adaptivity mechanisms. The examples of multi robotic frameworks are yet in the early stage of development. The problems with integration of single robot, multirobot, and machine learning concepts lie in the relatively low level of abstraction, with which the autonomous robots are currently modeled.

# References


[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[2] Rachid Alami and Silvia Silva da Costa Bothelho. Plan-based multirobot cooperation. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents,*, pages 1–20. Springer-Verlag, 2002.

[3] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1), January 2003.

[4] Rajeev Alur, Aveek K. Das, Joel M. Esposito, Rafael B. Fierro, Gregory Z. Grudic, Yerang Hur, Vijay Kumar, Insup Lee, J. P. Lee, James P. Ostrowski, George J. Pappas, B. Southall, J. Spletzer, and C. J. Taylor. A framework and architecture for multirobot coordination. In *Experimental Robotics VII*, pages 303–312. Springer-Verlag, 2001.

[5] R. C. Arkin. Towards the unification of navigational planning and reactive control. In *Working Notes of the AAAI Spring Symp. on Robot Navigatio*, pages 1–5, Stanford, USA, March 1989.

[6] R. C. Arkin, E. M. Riseman, and A. R. Hanson. AuRA: An architecture for vision-based robot navigation. In *Proc. of the Image Understanding Workshop*, pages 417–431, Los Angeles, CA, February 1987.

[7] R. Axelrod. *The Evolution of Cooperation*. HarperCollins, 1984.

[8] Geoffrey Biggs and Bruce MacDonald. A survey of robot programming systems. In *Proceedings of the Australasian Conference on Robotics and Automation*, CSIRO, Brisbane, Australia, December 1–3 2003.

[9] Stephanie Bodoff, Dale Green, Kim Haase, Eric Jendrock, Monica Pawlan, and Beth Stearns. *The J2EE tutorial*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[10] R. P. Bonasso. Integrating reaction plans and layered competences through synchronous control. In *Proc. of the 12th IJCAI*, pages 1225–1231, Sidney, Australia, 1991.

[11] R. Peter Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3):237–256, 1997.

[12] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: Background, model, and theory. In *Proceedings of the 33rd Conference on Decision and Control*, Control Systems Society, pages 4228–4234. IEEE, December 1994.

[13] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2, No1:14–23, 1986.

[14] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. Technical Report 1091, MIT AI Lab, February 1989.

[15] Rodney A. Brooks. Elephants don't play chess. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 3–15. MIT Press, 1990.

[16] Rodney A. Brooks. Intelligence without representation. *Artif. Intell.*, 47(1-3):139–159, 1991.

[17] Rodney A. Brooks. New approaches to robotics. *Science*, 253:1227–1232, 1991.

[18] Herman Bruyninckx. OROCOS applications. http://www.orocos.org/, 2003.

[19] Herman Bruyninckx, Peter Soetens, and Bob Koninckx. A software framework for component-based distributed feedback control kernels. http://www.orocos.org/, 2004.

[20] Joanna Bryson and Lynn Andrea Stein. Modularity and specialized learning in the organization of behaviour. In Robert French and Jacques Sougné, editors, *The Sixth Neural Computation and Psychology Workshop (NCPW6)*. Springer-Verlag: Heidelberg, Germany, 2000.

[21] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[22] Y. Uny Cao, A. Fukunaga, A. Kahng, and F. Meng. Cooperative mobile robotics: antecedents and directions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*, pages 226–234, Pittsburgh PA, USA, 1995.

[23] Ole Caprani, Jakob Fredslund, Jens Jacobsen, Line Kramhøft, Rasmus B. Lunding, Jørgen Møller Ilsøe, and Mads Wahlberg. *Evolution of Computer Bugs - an Interdisciplinary Team Work*. K.H.Madsen, 2002. Chapter in fortcomming book.

[24] L. Chaimowicz, T. Sugar, V. Kumar, and M. Campos. An architecture for tightly coupled multi-robot cooperation. In *International Conference on Robotics and Automation*, Seoul, Korea, May 2001.

[25] Bradley J. Clement and Anthony C. Barrett. Continual coordination through shared activities. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 57–64. ACM Press, 2003.

[26] Jonathan H. Connell. Creature building with the subsumption architecture. In *IJCAI-97*, pages 1124–1126, Milan, August 1987.

[27] Jonathan H. Connell. A colony architecture for an artificial creature. Technical Report 1151, Massachusets Institute of Technology Artificial Inteligence Laboratory, 1989.

[28] Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2719–2724, Nice, France, 1991.

[29] Randall Davis and Reid G. Smith. Negotiation distributed as a metaphor for problem solving. *Communication in Multiagent Systems*, pages 51–97, 2003.

[30] A. Deshpande, A. Goellue, and P. Varaiya. SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. *Lecture Notes in Computer Science*, 1273:113–133, 1997.

[31] M Bernardine Dias and Anthony (Tony) Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, July 2000.

[32] M Bernardine Dias and Anthony (Tony) Stentz. A market approach to multirobot coordination. Technical Report CMU-RI -TR-01-26, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2001.

[33] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[34] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for multi-agent systems. *Autonomous Robots*, 3:375–397, 1996.

[35] E. H. Durfee and T. A. Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In *Proc. of AAAI-90*, pages 86–93, Boston, MA, 1990.

[36] Rachid Alami Félix Ingrand, Raja Chatila. An architecture for dependable autonomous robots. In *IARP-IEEE RAS Workshop on Dependable Robotics*, Seoul, South Korea, May 2001.

[37] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 88–97. Kaufmann, San Mateo, CA, 1990.

[38] Robert James Firby. *Adaptive execution in complex dynamic worlds*. PhD thesis, Yale University Computer Science Department, 1989.

[39] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3508–3514, San Diego, California, May 1994.

[40] M. Fox and D. Long. Pddl2.1: An extension of pddl for expressing temporal planning domains. *Journal of AI Research*, 20:61–124, 2003.

[41] J.A. Franklin. Refinement of robot motor skills through reinforcement learning. In *Proceedings of 27th IEEE conference on Decision and Control*, pages 1096–1101, TX, Austin, 1988.

[42] Erann Gat. ALFA: A language for programming reactive robotic control systems. In *IEEE International Conference on Robotics and Automation*, pages 1116–1121, 1991.

[43] Erann Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic Inst. & State Univ., Blacksburg, 1991.

[44] Erann Gat. Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots. In *In Proc. of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 809–815, 1992.

[45] Erann Gat. Esl: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the IEEE Aerospace Conference*, 1997.

[46] Erann Gat. On three-layer architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, 1997.

[47] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Proceedings of AAAI-87*, pages 677–682, 1987.

[48] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Pub Co, January 1989.

[49] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artif. Intell.*, 86(2):269–357, 1996.

[50] Ralph Hartley and Frank Pipitone. Experiments with the subsumption architecture. In *International Conference on Robotics and Automation (ICRA)*, 1991.

[51] Inman Harvey, Philip Husbands, and Dave Cliff. Issues in evolutionary robotics. In *Proceedings of the second international conference on From animals to animats 2 : simulation of adaptive behavior*, pages 364–373. MIT Press, 1993.

[52] J. Hoff and G. Bekey. An architecture for behavior coordination learning. In *IEEE International Conference on Neural Networks*, Australia, 1995.

[53] M. Humphrys. W-learning: Competition among selfish q-learners. Technical Report 362, Computer Laboratory, University of Cambridge, 1995.

[54] IEEE Design Automation Standards Committee (DASC). *VHDL 1076.1 Language Reference Manual*, March 1999. http://www.eda.org/vhdl-ams.

[55] F. Ingrand, R. Chatila, R. Alami, and F. Robert. Prs: A high level supervision and control language for autonomous mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, St Paul, USA, 1996.

[56] James Jennings and Chris Kirkwood-Watts. Distributed mobile robotics by the method of dynamic teams. In *4th International Symposium on Distributed Autonomous Robotic Systems*, 1998.

[57] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: prospects, methods and challenges. *Int. J. of Group Decision and Negotiation*, 10(2):199–215, 2001.

[58] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In *Pattie Maes, editor, Designing Autonomous Agents*, pages 35–48. MIT Press, Cambridge (MA), 1990.

[59] Leslie Kaelbling. REX: A symbolic language for the design and parallel implementation of embedded systems. In *the AIAA Conference on Computers in Aerospace*, Wakefield, MA, 1987.

[60] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, 9(1):215–235, 1997.

[61] Thiemo Krink. Motivation networks - a biological model for autonomous agent control. *Journal of Theoretical Biology*, 2000.

[62] J. E. Laird and P. S. Rosenbloom. Integrating execution, planning, and learning in soar for external environments. In P. S. Rosenbloom, J. E. Laird, and A. Newell, editors, *The Soar Papers: Research on Integrated Intelligence (Volume 2)*, pages 1036–1043. MIT Press, London, 1993.

[63] Edward A. Lee. *Overview of the Ptolemy Project*. Technical Memorandum UCB/ERL M01/11, March 6 2001.

[64] M. Lindström, A. Orebäck, and H. Christensen. BERRA: A research architecture for service robots. In Khatib, editor, *Proceedings of the International Conference on Robotics and Automation*, San Francisco, May 2000.

[65] D.C. MacKenzie. *A Design Methodology for the Configuration of Behavior-Based Mobile Robots*. PhD thesis, Georgia Institute of Technology, College of Computing, 1997.

[66] P. Maes. How to do the right thing. *Connection Science Journal, Special Issue on Hybrid Systems*, 1, 1990.

[67] Pattie Maes. Modeling adaptive autonomous agents. *Artif. Life*, 1(1-2):135–162, 1994.

[68] Pattie Maes and Rodney A. Brooks. Learning to coordinate behaviors. In *National Conference on Artificial Intelligence*, pages 796–802, 1990.

[69] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pages 246–253. ACM Press, 2001.

[70] Jacek Malec. Behaviour-based autonomous systems: Towards an analysis framework. In *12th European Meeting on Cybernetics and Systems '94*, pages 1419–1426, Vienna, Austria, 1994. World Scientific.

[71] E. Martinson, A. Stoychev, and R. Arkin. Robot behavioral selection using q-learning. Technical Report GIT-CC-01-19, College of Computing, Georgia Institute of Technology, Atlanta, GA, 2001.

[72] Maja J. Matarić. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[73] Maja J. Matarić. Interaction and intelligent behavior. Technical report, MIT AI Lab Tech Report AITR-1495, Aug 1994.

[74] Maja J. Matarić. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189, Cambridge, Massachusetts, 1994.

[75] Maja J. Matarić. Issues and approaches in the design of collective autonomous agents. In *Robotics and Autonomous Systems*, volume 16, pages 321–331, Dec 1995.

[76] Maja J. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), 1997. special issue on Software Architectures for Physical Agents, Hexmoor, Horswill, and Kortenkamp, eds.

[77] S. E. Mattsson and M. Anderson. The ideas behind omola. In *CACSD 92: IEEE Symposium on Computer Aided Control System Design*, pages 23–29, 1992.

[78] S.E. Mattsson, H. Elmqvist, and M. Otter. Physical system modeling with modelica. In *Control Engineering Practice*, volume 6, pages 501–510, 1998.

[79] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. of AAAI-91*, pages 634–639, Anaheim, CA, 1991.

[80] David McAllester. Nonlinear strips planning. In *Lecture Notes for 6.824, Artificial Intelligence*, October 1993.

[81] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. Technical report, Department of Computer Science, Yale University, New Haven, CT, USA, 1998.

[82] Thomas Meyer, Norman Foo, Dongmo Zhang, and Rex Kwok. Logical foundations of negotiation: Outcome, concession and adaptation. In *Proceedings of AAAI04: Nineteenth National Conference on Artificial Intelligence*, 2004.

[83] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3rd edition, March 1996.

[84] Dana S. Nau, Yue Cao, Amnon Lotem, and Hector Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 968–975. Morgan Kaufmann Publishers Inc., 1999.

[85] Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, San Francisco, 1980.

[86] Nils J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA, 1984.

[87] Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[88] L. Parker. *Distributed Autonomous Robotic Systems*, volume 4, chapter Current State of the Art in Distributed Autonomous Mobile Robotics, pages 3–12. Springer-Verlag, Tokyo, 2000.

[89] L. E. Parker. Adaptive action selection for cooperative agent teams. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 442–450. MIT Press, 1992.

[90] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. In *IEEE Transactions on Robotics and Automation*, volume 14, pages 220–240, 1998.

[91] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.

[92] Mitchel Resnick. Beyond the centralized mindset. *Journal of the Learning Sciences*, 5(1):1–22, 1996.

[93] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34. ACM Press, 1987.

[94] F. Schönherr and J. Hertzberg. The DD&P robot control architecture. A preliminary report. In M. Beetz, J. Hertzberg, M. Ghallab, and M. Pollack, editors, *Advances in Plan-Based Control of Robotic Agents*, volume 2466 of *Lecture Notes in Artificial Intelligence*, pages 249–269. Springer, 2002.

[95] Reid Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings Conference on Intelligent Robotics and Systems*, October 1998.

[96] Reid Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. Sullivan. A layered architecture for office delivery robots. In *First International Conference on Autonomous Agents*, pages 235 – 242, February 1997.

[97] Reid Simmons, Sanjiv Singh, David Hershberger, Josue Ramos, and Trey Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In *Proc. of the ISER '00 Seventh International Symposium on Experimental Robotics*, December 2000.

[98] Reid Simmons, Trey Smith, M Bernardine Dias, Dani Goldberg, David Hershberger, Anthony (Tony) Stentz, and Robert Michael Zlot. A layered architecture for coordination of mobile robots. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Proceedings from the 2002 NRL Workshop on Multi-Robot Systems*. Kluwer Academic Publishers, May 2002.

[99] William D. Smart and Leslie Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA-2002)*, volume 4, pages 3404–3410, 2002.

[100] L. Spector. *Supervenience in Dynamic-World Planning*. PhD thesis, Department of Computer Science, University of Maryland, 1992.

[101] Anthony (Tony) Stentz and M Bernardine Dias. A free market architecture for coordinating multiple robots. Technical Report CMU-RI-TR-99-42, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1999.

[102] David B. Stewart and Pradeep Khosla. Rapid development of robotic applications using component-based real-time software. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 465–470, August 1995.

[103] Alexander Stoytchev and Ronald C. Arkin. Combining deliberation, reactivity, and motivation in the context of a behavior-based robot architecture. online papers, 2001.

[104] R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[105] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, March 1998.

[106] John D. Sweeney, TJ Brunette, Yunlei Yang, and Roderic A. Grupen. Coordinated teams of reactive mobile platforms. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2002.

[107] Clemens Szyperski. *Component software, beyond object-oriented programming.* Addison-Wesley, 1997.

[108] Yasutake Takahashi, Minoru Asada, and Koh Hosoda. Reasonable performance in less learning time by real robot based on incremental state space segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '96)*, pages 1518–1524, 1996.

[109] Ming Tan. Multi-agent reinforcement learning: independent vs. cooperative agents. *Readings in agents*, pages 487–494, 1998.

[110] J. H. Taylor. A modeling language for hybrid systems. In *Proceedings IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, pages 339–344, Tucson (AZ), USA, 7–9 1994.

[111] Georgios Theocharous, Khashayar Rohanimanesh, and Sridhar Mahadevan. Hierarchical learning and planning using multi-scale models. 2000.

[112] Régis Vincent, Pauline Berry, Andrew Agno, Charlie Ortiz, and David Wilkins. Teambotica: a robotic framework for integrated teaming, tasking, networking, and control. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1152–1153. ACM Press, 2003.

[113] C. J. C. H. Watkins. *Learning With Delayed Rewards.* PhD thesis, Cambrige University Psychology Department, 1989.

[114] Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 335–342, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.