Organic Computing

Dr. rer. nat. Christophe Bobda
Prof. Dr. Rolf Wanka
Department of Computer Science 12
Hardware-Software-Co-Design

Outline

- Organization and design of autonomous systems
  - Terminology and Concepts
- Architecture
  - Functional architecture
  - Operational architecture

Operational Architecture

Autonomous systems - operational Architecture

- In the previous section a general picture is given what capabilities an autonomous system
  - 6 examples of functional architectures give an overview the essential capabilities a mobile robot should possess.
- Most of the system design decisions have to be made at the operational level
- At this level the environmental constraints are put on the systems capabilities
- In this section, we will
  - describe the different operational constraints that have to be taken into account in an operational architecture.
  - illustration on one example: the operational architecture of the Autonomous Remote Agent at Deep Space 1

Autonomous systems - operational Architecture

- The operations meant are the elementary operations of the current virtual robot level
  - Elementary operations of a virtual robot are equivalent with the instructions of a virtual machine
- In this section we will
  - describe the appearance of elementary operations at different levels of abstraction,
  - how environmental constraints could be represented and
  - how the constraints could be coupled with the elementary operations of the different levels

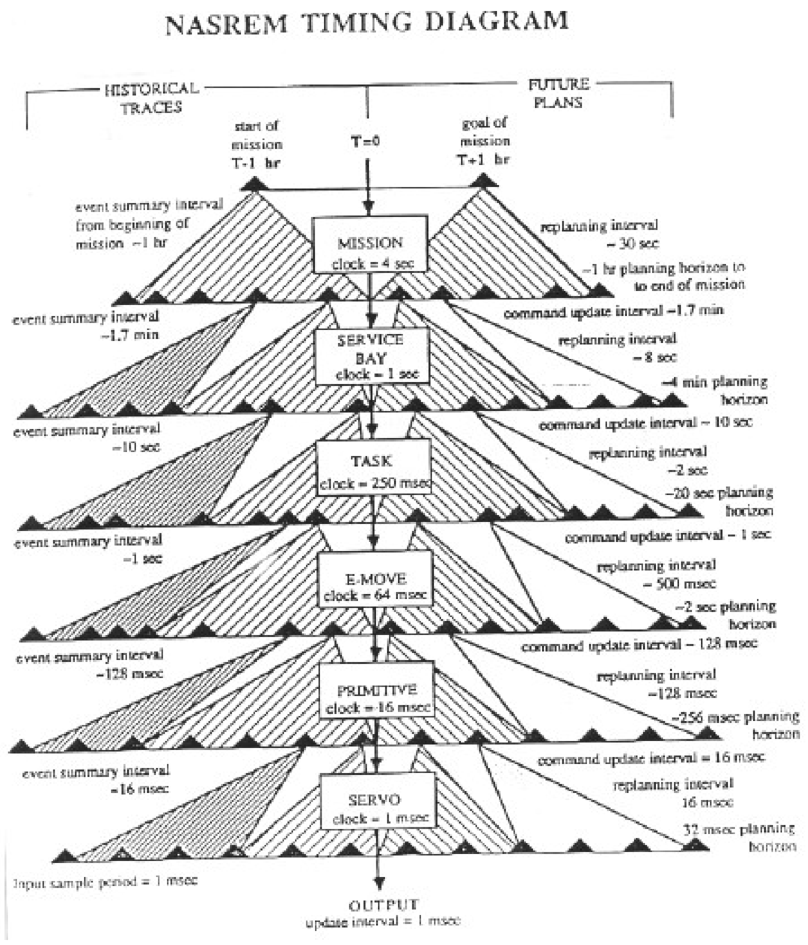
Operational Architecture – Time

- Very important and common constraint on autonomous systems
  - the time that is needed to perform an operation,
  - the time that is needed to plan an operation,
  - the time that is needed to acquire the information needed for an operation,
  - the time that is needed to verify the success of an operation
- The time constraint is much stronger at the lower levels of the autonomous system than at the higher levels of the system

Operational Architecture – Time

- Example: the relative timing between the six hierarchical layers of NASREM
- At the highest level, the complete backlog of the work for this mission is maintained, and the planning horizon is the end of the entire mission
- At each lower level, plans are formulated or selected to accomplish the instructions from the higher level
  - Each task in the higher level plan is decomposed into a lower level plan of lower level operations: subtasks
  - The planning horizon thus shrinks exponentially at each successively lower level of the hierarchy
  - the rate of operation completions increases at lower levels of the hierarchy, and decreases at the upper levels of the hierarchy
    - At the lowest (servo) level, the duration of an operation is fixed, one millisecond
    - At the higher levels the durations of the operations are variable
  - The higher in the hierarchy, the more pronounced this nonregularity becomes.

Operational Architecture – Time


NASREM TIMING DIAGRAM
HISTORICAL TRACES
FUTURE PLANS
start of mission T-1 hr
T=0
goal of mission T+1 hr
event summary interval from beginning of mission ~1 hr
MISSION clock = 4 sec
replanning interval ~30 sec
~1 hr planning horizon to to end of mission
event summary interval ~1.7 min
command update interval ~1.7 min
SERVICE BAY clock ~ 1 sec
replanning interval ~8 sec
~4 min planning horizon
event summary interval ~10 sec
command update interval ~ 10 sec
TASK clock ~ 250 msec
replanning interval ~2 sec
~20 sec planning horizon
event summary interval ~1 sec
command update interval ~ 1 sec
E-MOVE clock = 64 msec
replanning interval ~ 500 msec
~2 sec planning horizon
event summary interval ~128 msec
command update interval ~ 128 msec
PRIMITIVE clock = 16 msec
replanning interval ~128 msec
~256 msec planning horizon
event summary interval =16 msec
command update interval = 16 msec
SERVO clock = 1 msec
replanning interval 16 msec
32 msec planning horizon
Input sample period = 1 msec
OUTPUT update interval = 1 msec

Operational Architecture – Synchronization

- On a certain level the instructions from a higher level have to be decomposed into jobs for the different subsystems
- If there is only one subsystem available for a certain subtask, the assignment is not difficult
  - The remaining complexity is then in the dependencies between the different subtasks
- If more than one subsystem is available, communication between those subsystems will lead to the need to
  - synchronize the execution of the jobs at the different autonomous subsystems
- Synchronization constraints between concurrent operations can be given with the interval relationships of Allen
  - An interval a is represented by two events, its moment of initiation a’ and termination a"

Operational Architecture – Synchronization

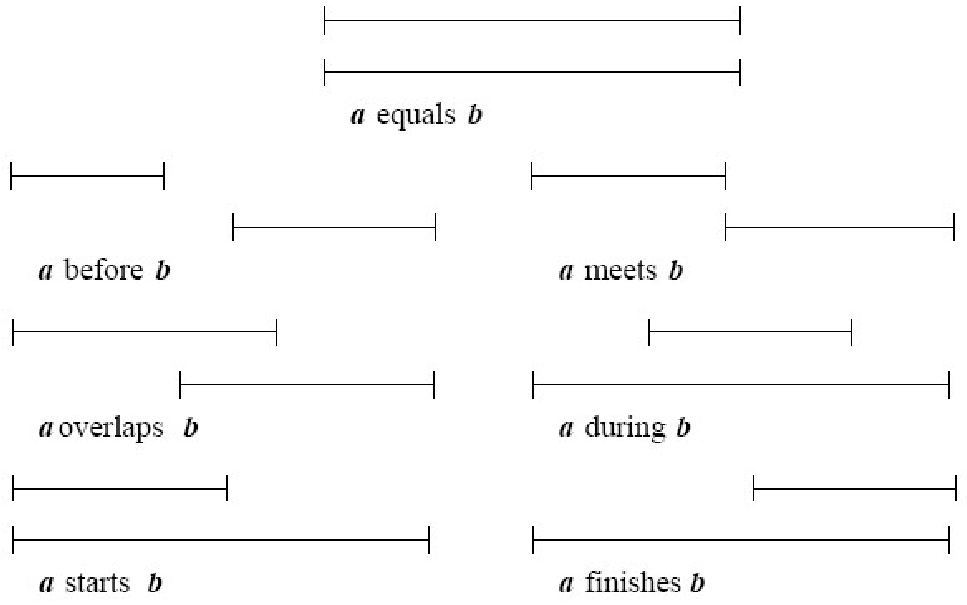
- The various possible relationships between two intervals a and b can then be described by the following relations between the initiation and termination events of both intervals

$$\begin{array}{lcl}
\mathbf{a}\ \text{equals}\ \mathbf{b} & \equiv & a' \rightleftharpoons b' \wedge a'' \rightleftharpoons b'' \\
\mathbf{a}\ \text{before}\ \mathbf{b} & \equiv & a'' \Rightarrow b' \\
\mathbf{a}\ \text{meets}\ \mathbf{b} & \equiv & a'' \rightleftharpoons b' \\
\mathbf{a}\ \text{overlaps}\ \mathbf{b} & \equiv & a' \Rightarrow b' \wedge a'' \Rightarrow b'' \\
\mathbf{a}\ \text{during}\ \mathbf{b} & \equiv & b' \Rightarrow a' \wedge a'' \Rightarrow b'' \\
\mathbf{a}\ \text{starts}\ \mathbf{b} & \equiv & a' \rightleftharpoons b' \wedge a'' \Rightarrow b'' \\
\mathbf{a}\ \text{finishes}\ \mathbf{b} & \equiv & b' \Rightarrow a' \wedge a'' \rightleftharpoons b''
\end{array}$$

- The operator $\Rightarrow$ Indicates a temporal order between two event and the operator $\rightleftharpoons$ Indicates simultaneity of two events

operational Architecture – Synchronization

- The interval relation between a and b can be reverted
  - For instance 'a equals b' is the reverse 'b equals a'
  - For the six other indicated relations the reverted relations are not equivalent with the original relations
    - The inverse of 'a before b' is for instance 'b after a',
    - The inverse of 'a during b' is for instance 'b contains a'.
    - In total 13 different relations between a and b can be indicated



Operational Architecture – Ordering

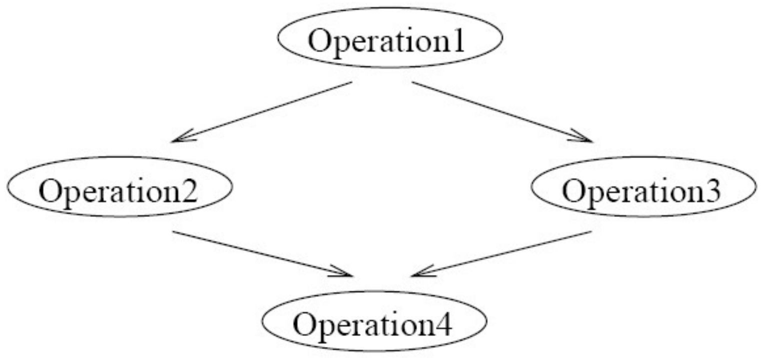
- If on the current level of the autonomous system only one subsystem or virtual machine is available to execute/interpret the instructions, the task given to this level can be decomposed into a set of operations for the lower layer
- When no alternative orders of execution are possible, the operation can be presented as a list or sequence
- When alternative orders of execution are still possible, the operations has to be presented as unordered set, with explicit orderings between those operations where such relation exist
- The advantage of the last approach is that the alternatives can be used to improve the robustness of the execution

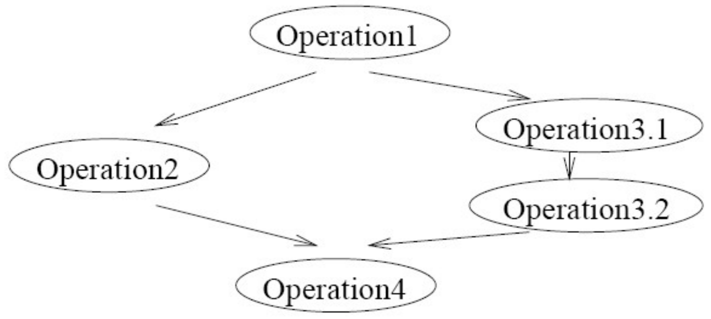
Operational Architecture – Ordering

- Because there is only one subsystem, no concurrency can take place (on this level).
- The only possible interval relations are before and after.
- A precedence graph is a visual aid to show this relationships
- A precedence graph is a directed acyclic graph, representing the ordering constraints between operations
- The precedence graph construct is a widely used representation of non-linear operations.
  - Notice that all operations in the graph have to be executed.

Operational Architecture – Ordering

- Set of ordering constraints
  - Operation1 before Operation2
  - Operation1 before Operation3
  - Operation2 before Operation4
  - Operation3 before Operation4
  - two possible sequences
    - (1, 2, 3, 4) and (1, 3, 2, 4)
  - splitting in the graph doesn't represent if-then-else alternatives, a splitting only represents alternative orders
  - Complexity of scheduling can be prevented by hiding the details of an operation





Operational Architecture – Tasks and subtasks

- A task represents the work to be done, or an activity to be performed
- A task is a required change in the world that has to be performed by the (autonomous) system that is assigned to this task
  - The system itself is a part of the world
- A task consists typically of an activity which begin is marked with an initiation event, and its end with a termination event
- The state of the world at the initiation event is the initial state, the state at the termination event the final state
- The final state has to be equivalent with a 'goal' state for all relevant state variables.
- The difference between the initial and goal state is the change the task has to accomplish

Operational Architecture – Tasks and subtasks

- Another possibility to specify a task is to do it with the reference to an abstract activity
- The term activity is used here as generic term for a single operation, a set or sequence of operations, or even as a conjunction of a to-be-planned number of operations at an unspecified number of abstraction levels
  - The only constraint to be impose is that the specification of the abstract activity contains enough information to guess the precise instantiation of the operations to perform the task
- Example: mobile robot. A possible specification would be to request the activity 'drive 1 meter'
  - We assume a very simple world model, just consisting of the robots Cartesian position (x; y) and its orientation OE, so x = (x; y; OE)T .
    - In the initial state the robot would be at position (x; y) = (0m; 0m), with a heading of OE = 0^o
    - The final position will be (x; y) = (0m; 1m)

Operational Architecture – Tasks and subtasks

- An alternative specification of the task would be a request for the state change ‘y = 0m => y = 1m’
  - This assumes that all other variables are not affected by this transformation
- Implicit description of the task describes only what hast to be done and how
  - ‘Drive 1 meter’
- An explicit description of a task translates the implicit description into a number of operations the robot could execute
  - This set of operations is called a fully instantiated task plan
  - The generation a task plan is typically started by a search through a library of activities that the system can perform
  - They search-key can be the requested state-change, or the abstract activity description
  - The result of a successful search is a task plan or a task frame

Operational Architecture – Tasks and subtasks

- A task plan is instantiated, but doesn't have to be fully instantiated
- Both a task frame as a task plan can consist of several partial plans
  - abstract descriptions of what have to be done
- The description of a partial plan can be used as a new key for a new iteration of the task planner
- In case of non successful search through the library of activities, the goal can be separated into a number of sub goals.
  - If for instance the requested state change involves more than one state variables, an attempt can be made to decouple the transformations.
- A complex state change can then be decomposed in that way into a set of simpler state changes.
- A search for such a simple state change is often successful

Operational Architecture – Tasks and subtasks

- If the goal couldn't be decomposed, an attempt can be made to find a decomposition by doing for instance a forward state space search.
  - For the initial state the set of possible actions is requested, the most promising action is applied (in simulation, so the activity library most also contain the expected result of the activity)
  - depending if there is a metric to judge if the resulting state is closer or further from the goal state, the search is continued depth-first or a breadth-first manner.
  - Backward search from the goal state is of course also possible.
- A task plan can be represented in different ways, depending on the algorithm that is used in the generation process.
  - If a fully instantiated task plan is produced in several hierarchical steps, a tree representation is convenient
  - An alternative representation is an AND/OR graph. of the task

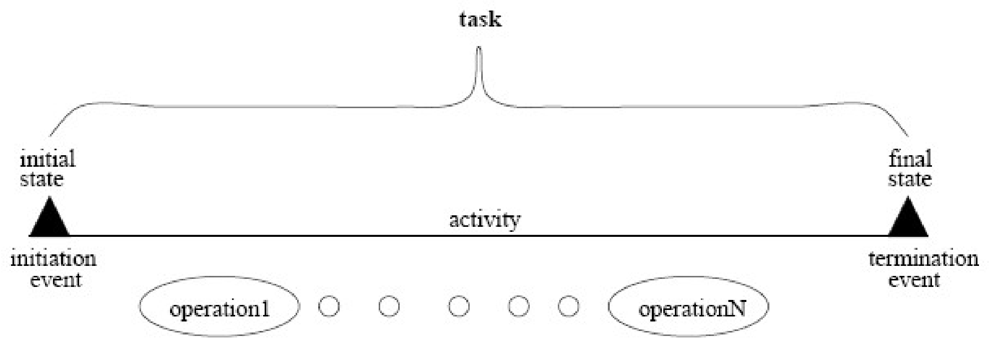
Operational Architecture – Tasks and subtasks

- Task Tree:
  - A task tree is an ordered tree with the following properties:
    - the root node represents the task
    - all other nodes represent a subtask
    - the (sub)task is satisfied when all the subtasks represented by the siblings are satisfied
    - the left to right order of siblings of a node represents the temporal order of the fulfillment of the subtask represented by the siblings
  - A task tree in which the leave represents the elementary operations of the autonomous system at a certain level is called a task decomposition

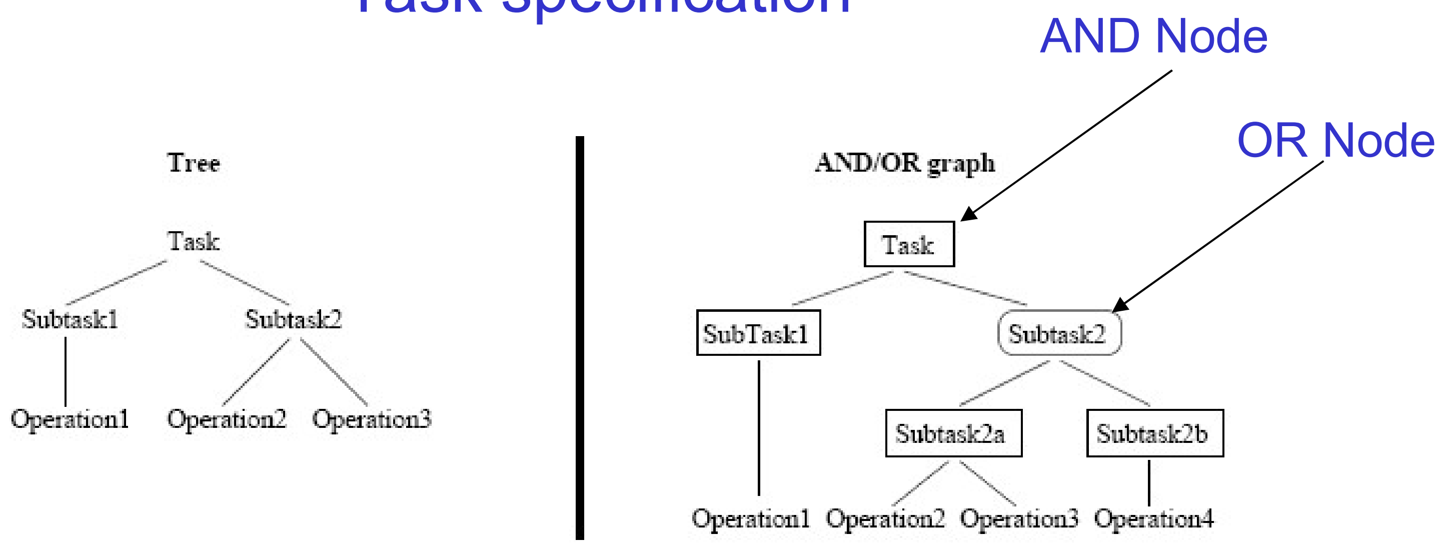
Operational Architecture – Tasks and subtasks

- An AND/OR graph can be used to represent alternatives partial plans to satisfy a subgoal of the task.
- The definition of an AND/OR graph is equivalent with those of a task tree, but two types of nodes are distinguished.
  - An AND-node represents a necessary connection. It indicates that all siblings have to be executed in the indicated order.
    - Only when all siblings satisfy their subgoal, the (sub)goal of the current node is satisfied.
  - The OR-nodes represent alternative connections. It indicates that only one sibling has to satisfy its subgoal to satisfy the (sub)goal of the current node.
    - The order of the siblings indicates the order in which the alternatives are tried.

Operational Architecture – Tasks and subtasks



Task specification



Task representation

Operational Architecture – binding

- Strong binding
  - If the environmental conditions change, the original plan is adjusted, but the impact of this changes on the original plan is as limited as possible
- Weak Binding
  - Restrict the a-priori planning to the definition of a set of planning instructions, which can be applied in certain environmental conditions, foreseen for the intended mission.
    - If the actual conditions correspond to the expectations, the planning instructions are performed, and the resulting list of operation are directly executed.
- A trade-off is necessary between Strong and Weak Binding is preferable
  - In a structured environment, the behavior of plan-driven control architecture can be prepared, and a plan can become mature by frequent use
  - In unknown environments the Weak Binding is favorite.

Operational Architecture – Interruption and exceptions

- Most behavior in the real world (either by humans or robots) is fraught by with uncertainty
- Actions can fail to have their expected effect, a plan can work thousands of times and then suddenly fail
- Given complete knowledge of the world this faults would not occur, but in general the uncertainty cannot be removed
- Humans are usually quite capable of finding the right balance between uncertainty and the effort of acquiring additional information
- A planning system that takes uncertainty into account in this natural way does not exist
- Many causes can interrupt the normal execution of a task plan
- If no precompiled response to the interrupt is available, we call the interrupt an exception

Operational Architecture – Interruption and exceptions

- On exception the plan generation system has to be activated, which has to replan the task
- Three type of errors can occur during execution :
  - software-errors, hardware errors, and external-errors
  - Software errors are associated with logic programming errors in the control programs (e.g.: endless loops, division by zero)
  - Hardware errors result from a malfunction of the hardware (power supply failures, sensor malfunction or actuator breakdown)
  - Soft- and hardware errors are also called internal-errors of the system
  - External errors are due to a discrepancy between the assumed and the real condition of the environment around the autonomous system
  - External errors can be divided in informational and operational errors

Operational Architecture – Interruption and exceptions

- An informational error is due to a difference between the internal model description of the environment and the description derived from sensor information
  - Informational errors do not necessarily have to lead to an exception. They merely indicate that some corrective action is needed.
- Operational errors are due to some physical unanticipated change of the environment
  - Collisions, grasp errors, part slippage, and tool breakdown
- The occurrence (and detection) of an error can lead to an interruption of the nominal flow of operations through the system
  - As the interrupt can not be handled on the current virtual robot layer, an exception message is send to the virtual robot layer above
  - as long as the situation cannot be handled, an exception is encountered at several virtual robot layers

Operational Architecture – Interruption and exceptions

- An autonomous system must be able to handle as much as possible situations
- This means that concurrent with the execution of the planned task the environment has to be monitored
- Because of the concurrent character of the monitoring, the monitoring processes have to be as lightweighted as possible
- Initially it is enough to know that the situation is different from expected
  - As soon as this is detected, the reasoning about the actual situation, and how to handle the situation, can start

Operational Architecture – Interruption and exceptions

- Detection of exceptions
  - Some errors that occur during the execution of an activity can hardly be missed.
    - No special effort has to be made for instance to detect that your mobile robot hit a tree
  - The occurrence of obvious errors can generally be predicted from earlier measurements
    - corrective actions can be used to prevented the system breakdown
- Several issues must be distinguished in detecting exeption
  - In the first place the detection of an exception is not necessarily the same as the detection of the cause of the exception.
    - In complex systems a deviation from planned behavior can lead to an exception at a total distinct place. Both distinct in space as in time
  - the monitor should primarily detect the characteristics of an exception and not the cause
  - The monitor passes the information of a detected exception as fast as possible to the other control functions supervising the process

Operational Architecture – Interruption and exceptions

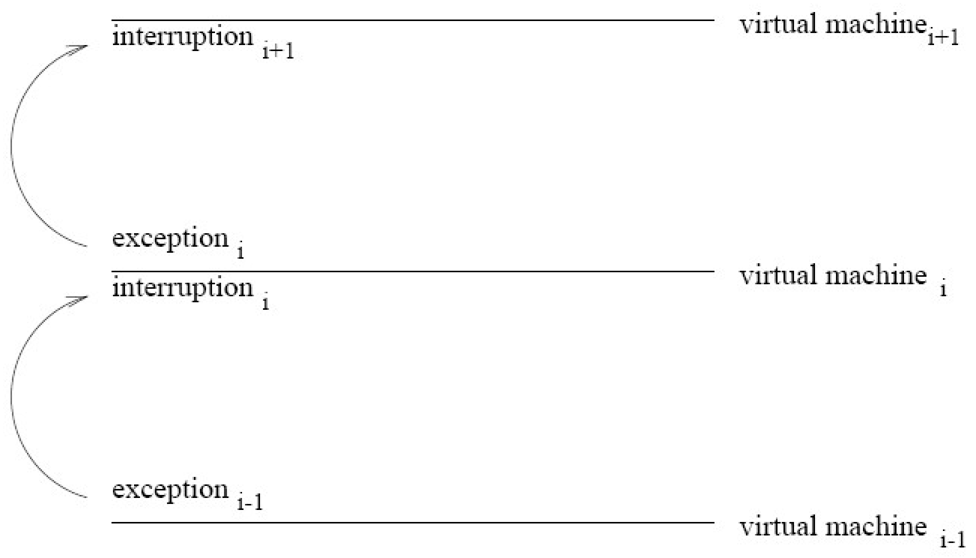
- The second distinction is related to the moment when a breakdown is detected and reported.
- In the ideal case the monitor is able to predict the breakdown of a process before it actually takes place.
  - A breakdown is mostly not an event which comes without announcement
  - A small error can lead to a gradual degradation of the process performance until finally the process breaks down
- Monitoring involves the feature measurements
  - The better these features give information on the small errors, the earlier the monitor is able to detect the exception
- An important prerequisite is that a model is available describing the relation between small errors and serious ones.
- Such a model contains parameters with which the behavior of the system is determined
- Minimal n tests have to be made monitor a process described with a model with n parameters

Operational Architecture – Interruption and exceptions

- The monitor activity can be described by the set of conditions to be checked
- A major problem with the application of such models and related monitor conditions is that the parameters of the model are not always observable
- The parameters in a model are chosen for their correspondence with a physical entity and not for their correspondence with some sensor measurement
- A transformation is needed between the measurable parameters (also called the observables) and the internal parameters of the model
- Extensive research in this area has been performed and various parameter estimation techniques (Kalman filtering for instance) have been applied
- Once such transformation is found, the monitor conditions can be expressed directly in the observables.

Operational Architecture – Interruption and exceptions

- The problem can also be approached from an other way.
- Taking each activity that is to be performed, a list of possible exceptions is made
- This list is compared with the list of available virtual sensor, and an analysis is made what correspondence could be found between the sensor signals and occurrence of exceptions



Exception at lower level are interrupts at higher level

Operational Architecture – Interruption and exceptions

- Handling of exceptions: Before any corrective action can be applied, the current situation has to be analyzed
  - Detection and handling of exceptions can be classified as the diagnostic capabilities of the system
    - Self healing capability
  - The information already available (i.e. backlog, other sensors) has to be searched for a hypothesis about the precise cause for the occurrence of the exception
  - Verification plans can be executed to distinct several possible classes of exceptions
  - A verification plan can contain several advanced sensing activities
  - Sensing could also provide the latest information about the world
  - An up-to-date world can be very valuable at this moment
    - parts of the current task plan have to be replanned on this data

Operational Architecture – Interruption and exceptions

- If the exception was caused by an information error, there is a change that the values of the other parameters of the world model are also wrong
- The most straight forward method is to inspect all information in the world model
  - this can be very time consuming
- In practical cases the classification of the exception is used to estimate which entities in the world model have a high probability to be changed
- No general methodology is found for this approach yet
- The planning of a recovery activity is conceptually equivalent with the planning of the activities of the task itself
  - The only difference is that the knowledge of the exceptional situation can be used to guide the planning process

Operational Architecture – Example

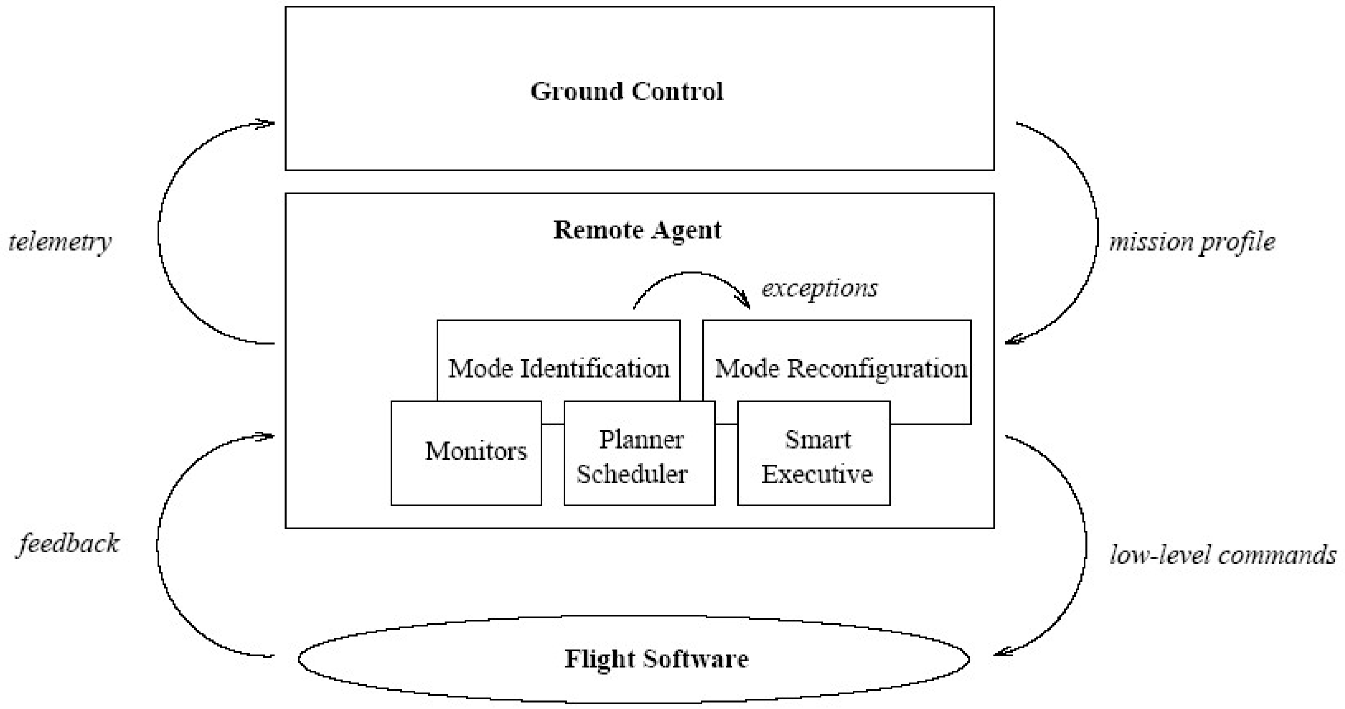
- The autonomous Remote Agent
  - one of the 12 technologies tested on Deep Space 1 (DS1)
    - spacecraft launched to especially test advanced technologies in space
- Good example for this section
  - Tight deadlines and resource constraints In space
  - no second chances stem from orbital dynamics and rare celestial events
  - Tight spacecraft resources, (renewable or non-renewable), must be carefully managed throughout the mission
  - The operational aspects play important role in the Remote Agent
- Goal: Control of the spacecraft for a long time
  - Two experiments: take the control over the spacecraft for respectively 6 hours and for 6 days
    - task normally performed by a ground crew up to 300 personnel

Operational Architecture – Example

- The remote agent functional model
  - The concurrent operation of the subsystems is coordinated by the Remote Agent
    - A spacecraft is complex system (flight computer, an on board processors connected to sophisticated sensors (e.g. star trackers), actuator subsystems (e.g. reaction wheels) and science instruments
  - The Remote Agent is a layer on top of the flight software.
  - The flight software is a virtual machine layer (complete real-time system)
    - Special functions for the on board hardware
  - The Smart EXEC is responsible for the control of the flight software
  - The Remote Agent is a layer below the Ground Control.
    - the spacecraft is instructed via the mission profile

Operational Architecture – Example

- Monitors are "virtual sensors", that pre-process complex data-streams so that they can be used in the Remote Agent



The functional architecture of the Remote Agent

Operational Architecture – Example

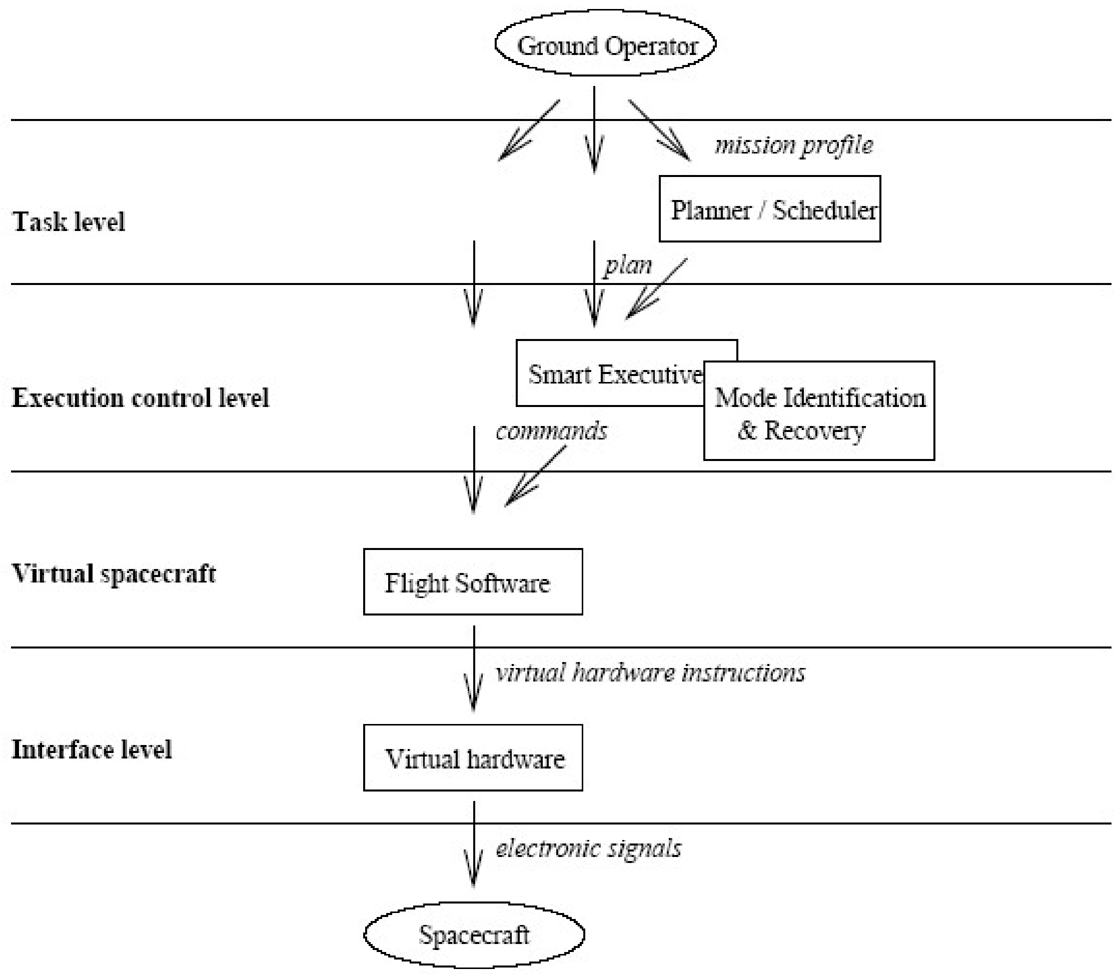
- The Planner/Scheduler (PS) can create plans that are more flexible and better able to take advantage of unexpected opportunities than plans created by ground controllers
- The Remote Agent must not only control the spacecraft, but also guarantee that the mission goals are successfully executed on time and with the minimum use of resources
- The reliability is guaranteed by a low-level fault protection system, the Mode Identification and Reconfiguration (MIR) system
  - constant monitoring of the spacecraft. Modelling of the bahavior of the spacecraft
  - Failure detection is based on conflicting expectation and sensor values
  - Type of failure determined by the sensor values
  - The failure is reported to the Smart Executive, which can ask the MIR for the best recovery action

Operational Architecture – Example

- The remote agent operational model
  - The execution control level is filled with two tightly coupled controllers
    - a procedural-based approach (EXEC), and
    - a deductive-based approach (MIR)
  - In practice the Smart Executive has the lead,
    - but in principle the MIR could control the spacecraft stand-alone
  - To control the spacecraft the MIR has to be fed with a number of goals (properties that have to be become true)
    - In theory this can be done by the ground operators
    - In practice the goals are only fed to the MIR by the Smart Executive in the case a plan fails
  - The Planner / Scheduler is one abstraction level higher than the execution control level

Operational Architecture – Example

Virtual machine levels for the Deep Space 1



Operational Architecture – Example

- The continuous operation is achieved by repetition of the following cycle:
- 1. Retrieve high level goals from the mission profile database
- 2. Ask the PS to generate a plan
  - The PS receives the goals, scheduling horizon and the state of all relevant spacecraft subsystems at the beginning of the scheduling horizon
  - The resulting plan is a set of tokens placed on various state variable time lines, with temporal constraints between tokens
- 3. Send the plan to the executive
  - The execution of the current plan continues. The execution of the new plan start on the beginning of the next scheduling horizon
  - Executes the commands, making sure that the commands succeed.
  - If not, it can create alternative command sequences or ask assistance from the MIR
  - When the recovery strategies run out of options, it will coordinate actions to bring the spacecraft in a "safe state" and request the advice of the PS

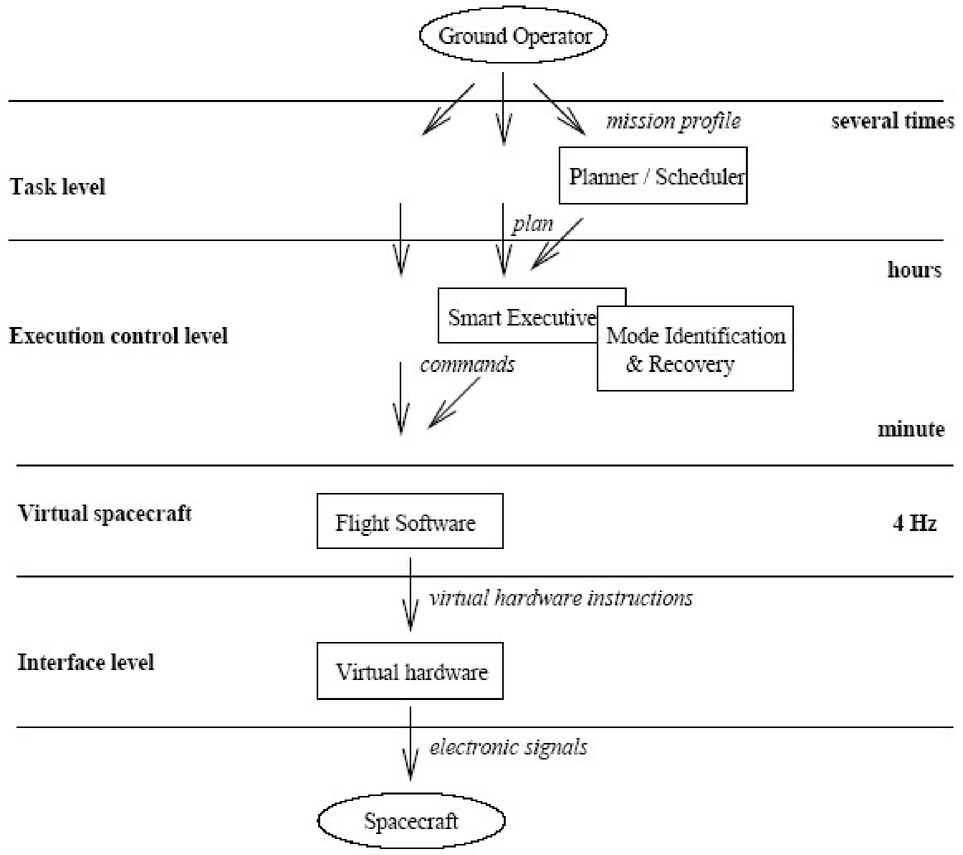
Operational Architecture – Example

- 4. Repeat the cycle from step 1 when:
  - Execution time has reached the end of the scheduling horizon (minus the time estimated for the planner to generate a new plan)
  - The executive has requested a new plan as a result of a hard failure
- Inside the smart executive the plan execution is straightforward
  - It gets a set of time-lines
    - Time-lines consist of a linear sequence of tokens, each of which represents an activity that should take place during a temporal period
  - A token has a start and end window, and a set of pre- and postconditions
  - In principle plan execution would be simply waiting for the start time of a token, check the pre-conditions, and execute the commands connected to the token
  - In practice the transition from one token to another token is not that smooth, because not all state-variables change their state instantaneously

Operational Architecture – Example

Time in the remote agent

- Platform
  - RAD6k a 20MHZ
  - VxWork OS
- Control lopp frequency
  - 4khz
- EXEC dispatcher is event based
- Plan generator
  - 8 hours



Operational Architecture – Example

- Interrupts and exceptions
  - Goal: make the Remote Agent as reliable as possible
  - Constant monitoring of spacecraft's state and comparison with predicted state
  - Discrepancies between predicted and monitored can mean that the MIR signals a failure to the EXEC
  - Not only failure detection, but also reasoning capabilities to predict the most reasonable source of the fault
  - Optimization
    - places the spacecraft in the least cost configuration that exhibits a desired behavior
  - Recovery
    - restores the spacecraft, by finding actions that repair failed components or by finding alternative ways of achieving goals
  - standby and safing
    - place the spacecraft in a safe state, in absence of recovery

Operational Architecture – Example

- Two main thrust engines in the spacecraft, both connected to the single fuel and oxygen tank
- The second engine is present for redundancy purposes only
  - switching between engines is nominally not done
  - Model of the different sort of valves, and their status present in the MIR
    - Dark valves are closed. Valves with a bar trough them are pyro-valves: can only be opened once with an explosive
    - Both configurations in figure satisfy the goal of providing thrust, with the left configuration is to be preferred (less pyro-valves open)
    - If circled valve fails, the right configuration can be used as alternative.