Timeline-based Planning and Execution: Theory and Practice

- PLATINUm -

A Novel framework for **PL**anning and **A**cting with **TI**meli**N**es under **U**ncertainty

Alessandro Umbrico

National Research Council of Italy (ISTC-CNR)

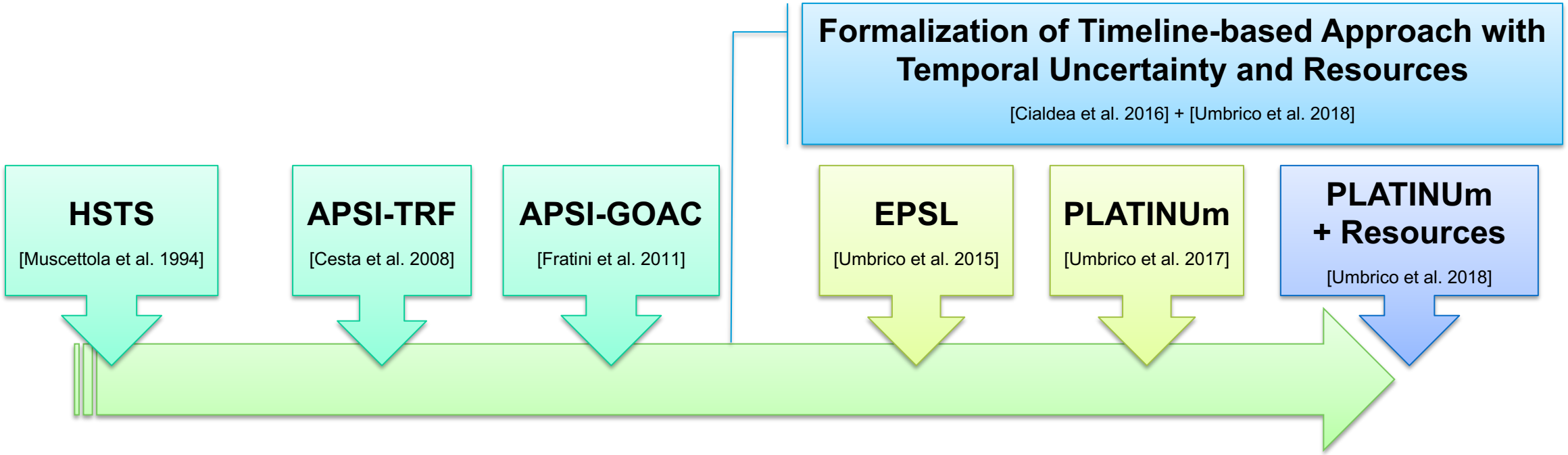
ISTC - CNR

Outline

- **General Introduction to PLATINUm**
  - History, general motivations and objectives
- **PLATINUm Representation Capabilities**
  - Temporal Uncertainty, Components and Resolvers
- **PLATINUm Deliberative Capabilities**
  - Pseudo-controllability aware planning and hierarchical approach
- **PLATINUm Executive Capabilities**
  - Closed-loop control approach and controllability issues
- **PLATINUm in Action**
  - Human-Robot Collaboration in realistic manufacturing scenarios

GENERAL INTRODUCTION TO PLATINUM

A Brief History of PLATINUm



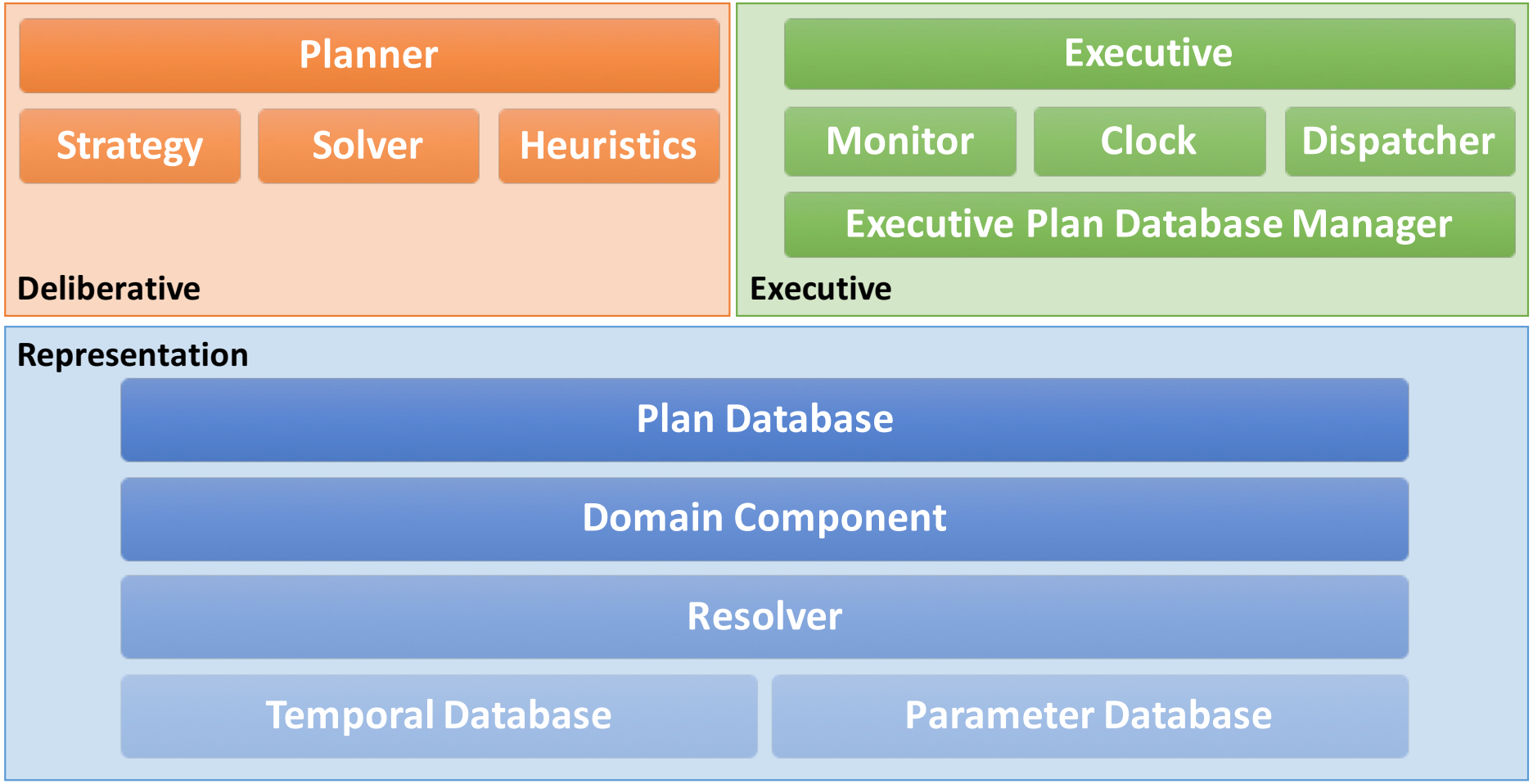
Motivations and Objectives

- Several timeline-based systems successfully applied in practice
  - E.g., **EUROPA2**, **ASPEN**, **IxTeT**, **APSI-TRF**
- Lack of **uniform formal interpretation** of the main concepts
  - E.g., different interpretation of timelines, domain rules, plans and solutions
- Lack of a **uniform approach** to **planning and execution** with timelines
  - E.g., different approaches to solving and modeling problems with timelines
- Lack of **representation of uncertainty** and uncontrollable dynamics

Framework Capabilities

- A software framework capable of dealing with planning and execution of timelines under temporal uncertainty
  - Comply with the **formalization proposed in [Cialdea et al. 2016]**
- Synthesize timeline-based plans with some desired **controllability properties**
  - E.g., **pseudo-controllability**
- Execute timelines by dealing with the **dynamics of the uncontrollable features of the environment**
  - **The controllability problem** [Morris, Muscettola and Vidal 2001]

Architectural Overview of the Framework

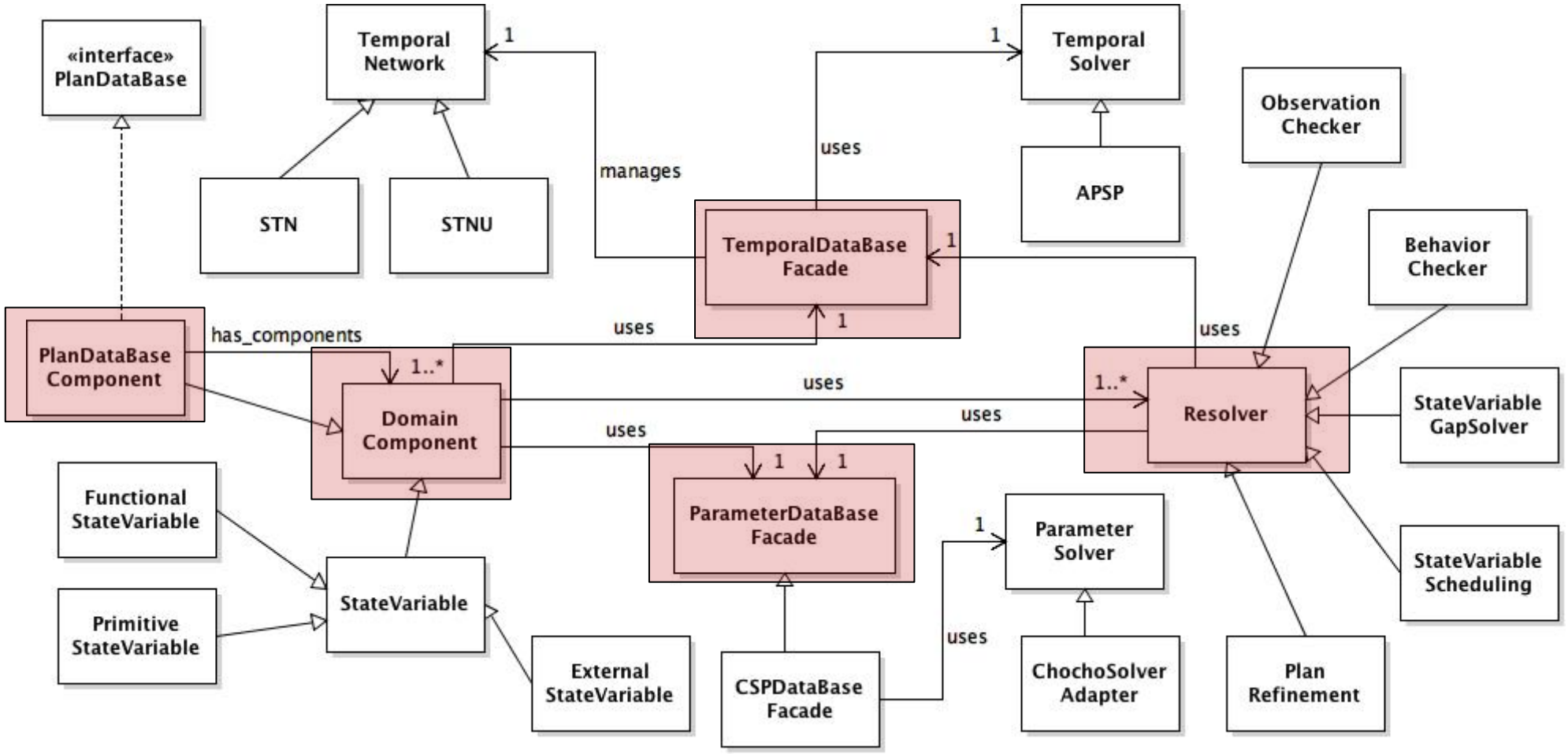


A Layered Architecture for PLATINUm

- The **Representation Layer** is responsible for providing the basic functionalities needed to manage timeline-based plans
  - Encapsulate data structures and algorithms needed to build valid timelines
- The **Deliberative Layer** is responsible for solving planning problems by synthesizing timeline-based solution plans
  - Encapsulate heuristics, strategies and algorithms needed to build plans
- The **Executive Layer** is responsible for executing timelines by scheduling flexible tokens over time
  - Encapsulate dispatching and monitoring algorithms to dynamically adapt timelines during execution, according to the received feedbacks

GOING DEEP INTO REPRESENTATION CAPABILITIES

Plan Database Overall Structure



Plan Data Representation and Management

- Temporal information is managed through **Temporal Networks**
  - Framework enabling temporal inference and consistency checking [Dechter et al. 1991]
- **Simple Temporal Network with Uncertainty** (STNU) to manage uncontrollable durations
    - **Controllability check** [Morris, Muscettola and Vidal 2001]
- A “standard” **CSP solver** is encapsulated to manage variable assignment and constraint propagation
  - Choco CSP solver

Resolvers and Components

- **Domain components** are data structures that model the features of a planning domain that must be controlled over time
  - Each feature has its own constraints that must be satisfied to generate valid timelines i.e., temporal behaviors without **flaws**
- **Resolvers** represent specialized algorithms capable of detecting flaws and computing possible solutions
  - Each resolver encapsulates the logic for handling a particular type of flaw
- PLATINUm provides a set of **ready to use components and resolvers** representing the typical features that compose a timeline-based planning domain

State Variables

- Components that comply with the proposed formalization
  - Encapsulate values, durations, controllability tags and transition constraints
- Resolvers are provided to synthesize timelines
  - **Schedule state variable tokens**
  - **Synthesize complete sequences of tokens** enforcing transition constraints

```java
public final class PrimitiveStateVariable extends StateVariable
{
    /**
     *
     * @param name
     */
    @DomainComponentConfiguration(resolvers = {
            // planning resolver
            ResolverType.PLAN_REFINEMENT,
            // scheduling resolver
            ResolverType.TIMELINE_SCHEDULING_RESOLVER,
            // time-line gap resolver
            ResolverType.TIMELINE_BEHAVIOR_PLANNING_RESOLVER,
    })
    protected PrimitiveStateVariable(String name) {
        super(name, DomainComponentType.SV_PRIMITIVE);
    }
```

```java
public final class TimelineSchedulingResolver extends Resolver<StateVariable>
{
    /**
     *
     */
    protected TimelineSchedulingResolver() {
        super(ResolverType.TIMELINE_SCHEDU
                ResolverType.TIMELINE_SCHE
    }

    /**
     *
     */
    @Override
    protected void doApply(FlawSolution so
            throws FlawSolutionApplicatior
    {
        // get the flaw solution to consid
        PrecedenceConstraint pc = (Precede
        try
        {
            // get reference and target de
            Decision reference = pc.getRef
            Decision target = pc.getTarget
```

```java
public final class TimelineBehaviorPlanningResolver extends Resolver<StateVariable>
    /**
     *
     */
    protected TimelineBehaviorPlanningResolver() {
        super(ResolverType.TIMELINE_BEHAVIOR_PLANNING_RESOLVER.getLabel(),
                ResolverType.TIMELINE_BEHAVIOR_PLANNING_RESOLVER.getFlawTypes());
    }

    /**
     *
     */
    @Override
    protected void doApply(FlawSolution solution)
            throws FlawSolutionApplicationException
    {
        // get the flaw solution to consider
        GapCompletion completion = (GapCompletion) solution;
        // check solution path
        if (completion.getPath().isEmpty())
        {
            try
            {
```

Discrete and Reservoir Resources

- Components **encapsulate resource constraints** and **resource events** according to the tokens of the timelines
- Resolvers provide the logic for detecting and solving peaks
  - Compute **pessimistic and optimistic resource profiles**
  - Compute peak solutions through Minimal Critical Sets (MCSs)
    - **Planning & Scheduling integration**

```java
public class ReservoirResource extends Resource
{
    public static final String CONSUMPTION_LABEL = "CONSUMPTION";
    public static final String PRODUCTION_LABEL = "PRODUCTION";
    private ResourceConsumptionValue consumption;
    private ResourceProductionValue production;

    /**
     *
     * @param name
     */
    @DomainComponentConfiguration(resolvers = {
            // planning resolver
            ResolverType.PLAN_REFINEMENT,
            // reservoir resource scheduler
            ResolverType.RESERVOIR_RESOURCE_SCHEDULING_RESOLVER
    })
    protected ReservoirResource(String name) {
        super(name, DomainComponentType.RESOURCE_RESERVOIR);
        this.min = Integer.MIN_VALUE + 1;
        this.max = Integer.MAX_VALUE - 1;
        this.initial = this.max;
        this.consumption = null;
        this.production = null;
    }
```

```java
public class DiscreteResource extends Resource
{
    public static final String REQUIREMENT_LABEL = "REQUIREMENT";
    private RequirementResourceValue requirement;

    /**
     *
     * @param name
     */
    @DomainComponentConfiguration(resolvers = {
            // planning resolver
            ResolverType.PLAN_REFINEMENT,
            // discrete resource scheduler
            ResolverType.DISCRETE_RESOURCE_SCHEDULING_RESOLVER
    })
    protected DiscreteResource(String name) {
        super(name, DomainComponentType.RESOURCE_DISCRETE);
        this.min = Integer.MIN_VALUE + 1;
        this.max = Integer.MAX_VALUE - 1;
        this.initial = this.max;
        // create the requirement value
        this.requirement = null;
```

```java
public class ReservoirResourceSchedulingResolver extends Resolver<ReservoirResource>
{
    /**
     *
     */
    protected ReservoirResourceSchedulingResolver() {
        super(ResolverType.RESERVOIR_RESOURCE_SCHEDULING_RESOLVER.getLabel(),
                ResolverType.RESERVOIR_RESOURCE_SCHEDULING_RESOLVER.getFlawTypes());
    }

    /**
     *
     */
    @Override
    protected List<Flaw> doFindFlaws()
    {
        // list of flaws
        List<Flaw> flaws = new ArrayList<>();
        try
        {
            // check pessimistic resource profile
            ReservoirResourceProfile prp = this.component.
                    computePessimisticResourceProfile();
            /*
             * Analyze the pessimistic profile and find peaks if
             * any and generate production checkpoints
             */
            flaws = this.doComputeProfilePeaks(prp);
```

```java
public class DiscreteResourceSchedulingResolver extends Resolver<DiscreteResource>
                implements Comparator<RequirementResourceProfileSample>
{
    /**
     *
     */
    protected DiscreteResourceSchedulingResolver() {
        super(ResolverType.DISCRETE_RESOURCE_SCHEDULING_RESOLVER.getLabel(),
                ResolverType.DISCRETE_RESOURCE_SCHEDULING_RESOLVER.getFlawTypes());
    }

    /**
     *
     */
    @Override
    protected List<Flaw> doFindFlaws()
    {
        // list of flaws
        List<Flaw> flaws = new ArrayList<>();
        try
        {
```

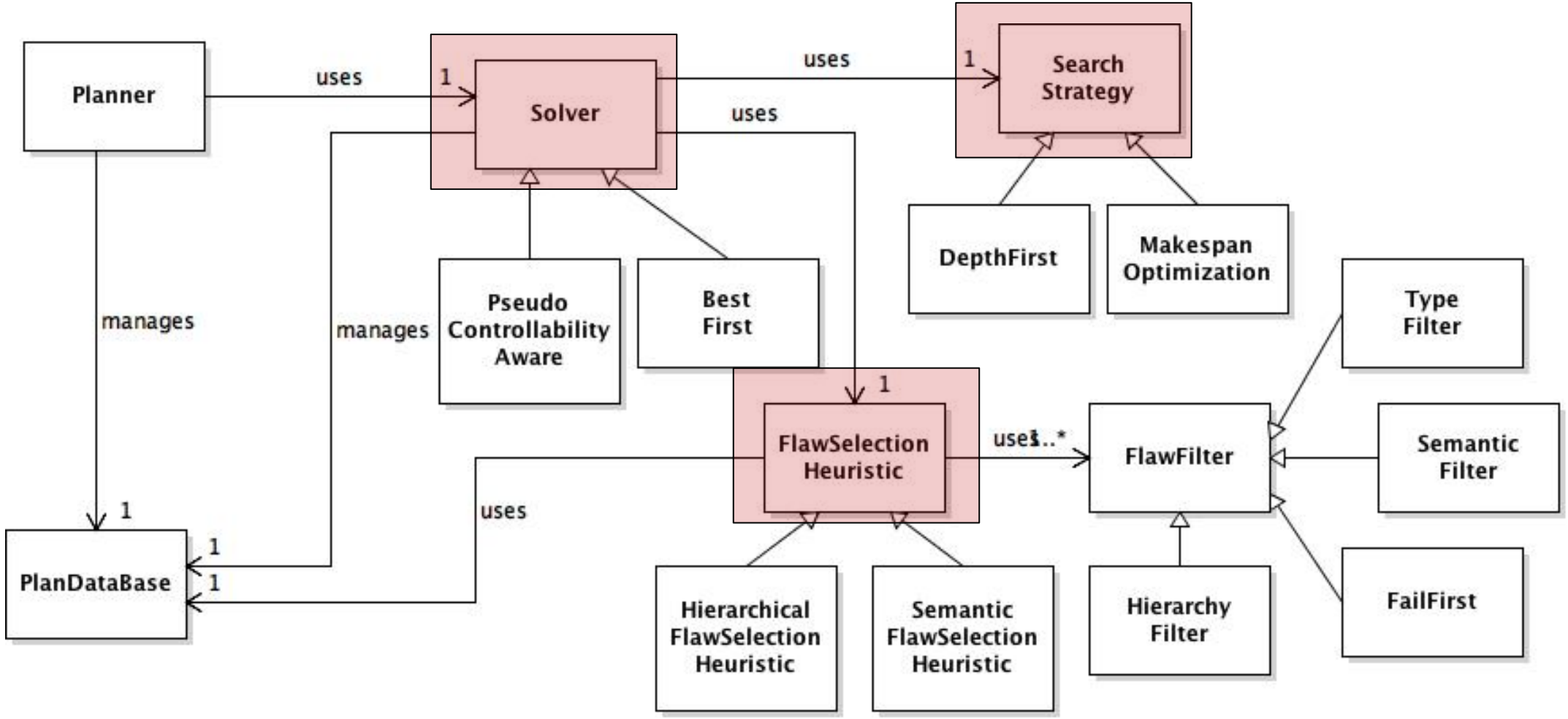
The Plan Database

- Encapsulating all the domain components and configurations
  - Composite design pattern
  - **Provide a public interface to domain features and data**
- Leverage internal components to detect **planning goals**
  - Planning goal solutions computed through **synchronization rules**
  - **Apply expansion or unification**

```
@DomainKnowledgeConfiguration(

        // set domain knowledge
        knowledge = DomainKnowledgeType.STATIC
)
@TemporalFacadeConfiguration(

        // set temporal network
        network = TemporalNetworkType.STNU,

        // set planner solver
        solver = TemporalSolverType.APSP
)
@ParameterFacadeConfiguration(

        // set parameter reasoner
        solver = ParameterSolverType.CHOCHO_SOLVER
)
@FrameworkLoggerConfiguration(

        // set logging level
        level = FrameworkLoggingLevel.DEBUG
)
public final class PlanDataBaseComponent extends DomainComponent implements PlanDataBase
{
    @DomainKnowledgePlaceholder
    protected DomainKnowledge knowledge;

    // see Composite design pattern
    private Map<String, DomainComponent> components;

    // the planning problem
    protected Problem problem;
```

GOING DEEP INTO DELIBERATIVE CAPABILITIES

Detailed Structure of a PLATINUm Planner

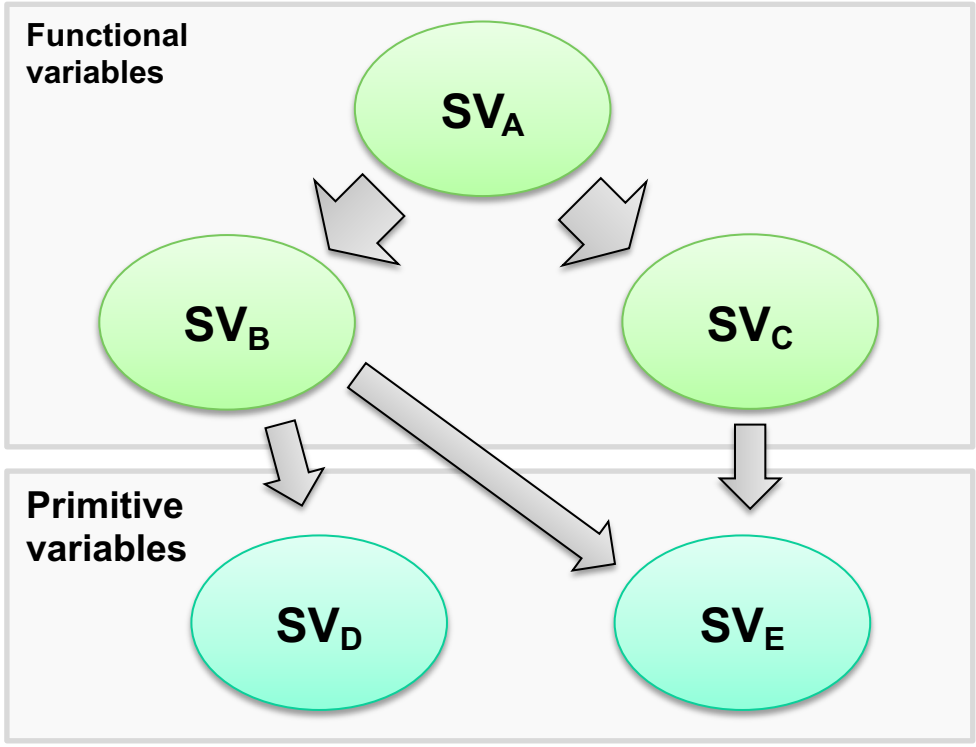


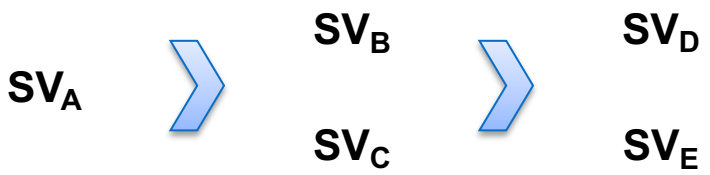
Pseudo-controllability Aware Solving

- General plan refinement search procedure
  - Iteratively refine an initial partial plan by solving flaws, until a valid plan without flaws is found
  - **Search decision point**: which partial plan to select for search space expansion
  - **Flaw decision point**: which flaw to solve for plan refinement
- **Pseudo-controllability check** as a special flaw of the plan
  - Verify if the flexible durations of the uncontrollable tokens have been modified with respect to the domain specification
  - **Pseudo-controllability is a necessary but not sufficient condition for dynamic controllability**

Hierarchical Flaw Selection Heuristics

- Analyze synchronization rules to extract **dependencies among state variables** and their values
  - Extract a **hierarchy** if possible
  - Domain-independent heuristics
- Leverage the extracted hierarchy to evaluate flaws and decide which flaw to solve next
  - **Support flaw decision point**





Search & Build Plan Synthesis

- **Search phase** aims at constraining temporal behaviors as much as possible
  - Interleave planning and scheduling decisions by reasoning on flaws
  - Constraint state variables behaviors according to synchronization rules and resource constraints - **generated behaviors are not timelines yet**
- **Build phase** aims at finalizing timelines by enforcing semantics defined by the formalization
  - **Synthesize valid timelines** according to the constrained behaviors of state variables generated by the search phase
  - Backtrack by **jumping-back** to the search phase in case of failures

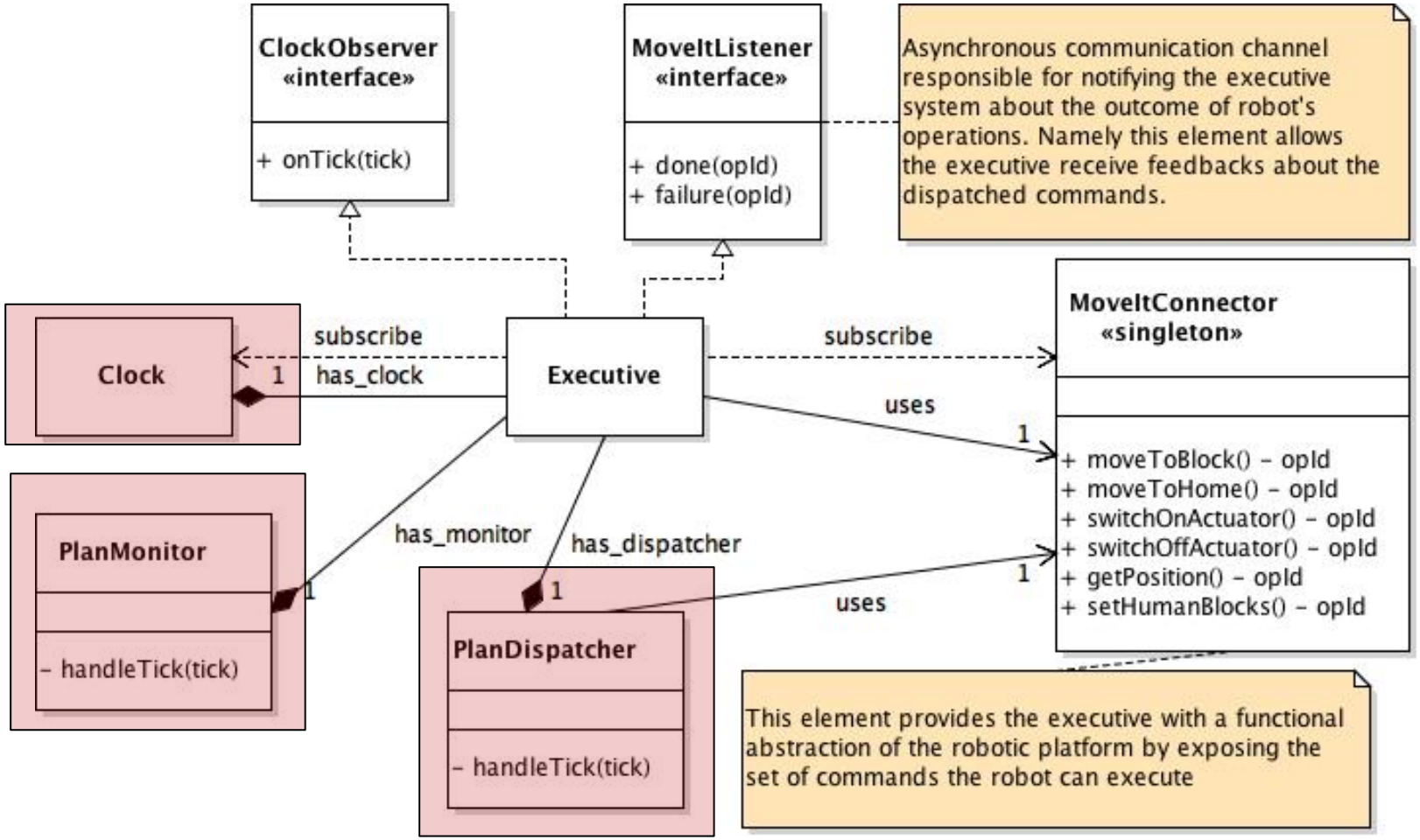
Defining and Using a PLATINUm Planner

```java
@PlannerSolverConfiguration(
        solver = PseudoControllabilityAwareSolver.class
)
@FlawSelectionHeuristicsConfiguration(
        heuristics = SearchAndBuildFlawSelectionHeuristic.class
)
@SearchStrategyConfiguration(
        strategy = DepthFirstSearchStrategy.class
)
@FrameworkLoggerConfiguration(
        level = FrameworkLoggingLevel.DEBUG
)
public class Planner extends DeliberativeObject
{
    @PlanDataBasePlaceholder
    protected PlanDataBase pdb;

    @PlannerSolverPlaceholder
    protected PlannerSolver solver;

    /**
     *
     */
    protected Planner() {}
```

```java
public static void main(String[] args)
{
    try
    {
        // build the plan database
        PlanDataBase pdb = PlanDataBaseBuilder.createAndSet(DDL, PDL);
        // initialize a planning instance of the plan database
        Planner planner = PlannerBuilder.createAndSet(pdb);

        // start planning
        SolutionPlan plan = planner.plan();
        // solution found
        System.out.println("... solution found after " + plan.getSolvingTime()
        // export plan encoding
        String enconding = plan.export();
        // print the resulting plan
        System.out.println("Resulting plan:\n" + plan + "\n");
        System.out.println("Exporting encoding:\n" + enconding + "\n");

        // display the resulting plant
        planner.display();
    }
    catch (NoSolutionFoundException ex) {
        // no solution found
        System.err.println(ex.getMessage());
    }
    catch (ProblemInitializationException ex) {
        System.err.println(ex.getMessage());
        ex.printStackTrace(System.err);
    }
    catch (Exception ex) {
        System.err.println(ex.getMessage());
        ex.printStackTrace(System.err);
    }
}
```

GOING DEEP INTO EXECUTIVE CAPABILITIES

Detailed Structure of a PLATINUm Executive

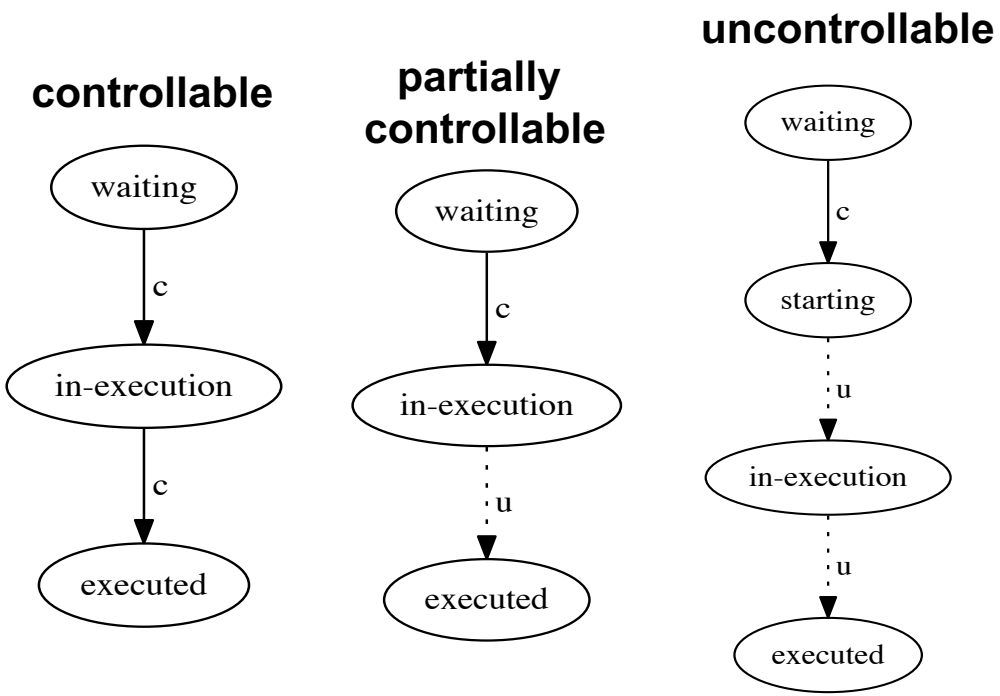


Closed-loop Execution of Timeline-based Plans

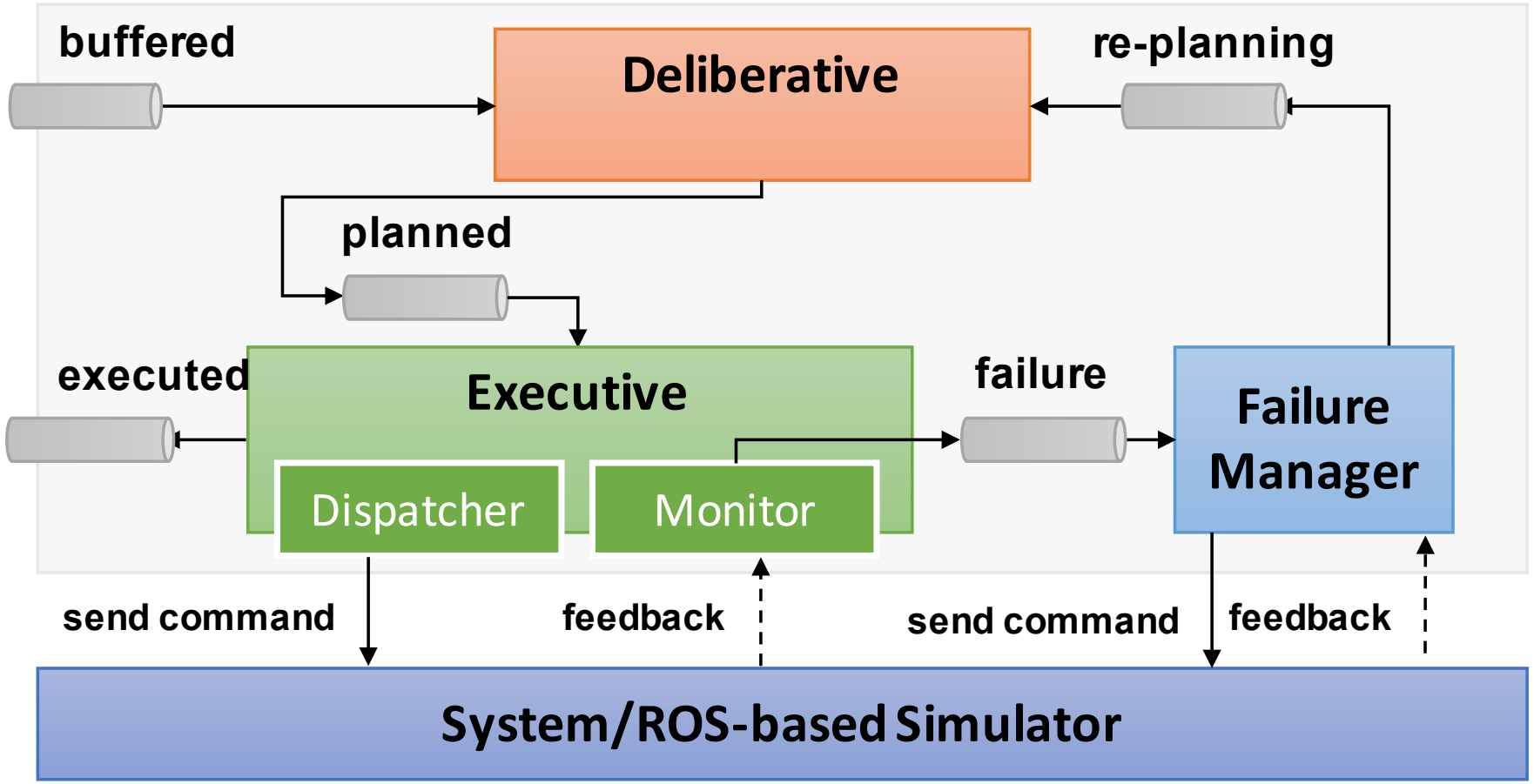
- A PLATINUm executive consists of a **closed-loop control process** which iteratively fix flexible timelines over time
- A **Dispatcher** actually executes the timelines of a plan by sending commands to a physical system
  - It is responsible for deciding the start of the execution of the tokens that compose the timelines of a plan
- A **Monitor** handles execution feedbacks to verify whether the plan complies with the observed status of the environment or not
  - It is responsible for propagating information about the actual duration of uncontrollable tokens

Executing Timelines under Temporal Uncertainty

- Extract **start/end execution dependencies** by analyzing temporal relations of a plan
  - Dynamically generate a **execution dependency graph**
- Manage token transitions according to their **controllability properties**
  - Different controllability properties entail different dispatching policies



Closed-loop Control Architecture



HUMAN-ROBOT COLLABORATION CASE STUDY

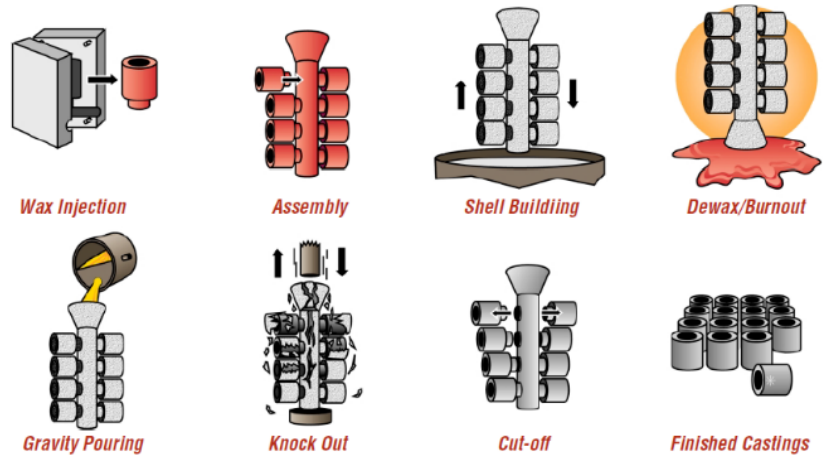
The FourByThree Research Project

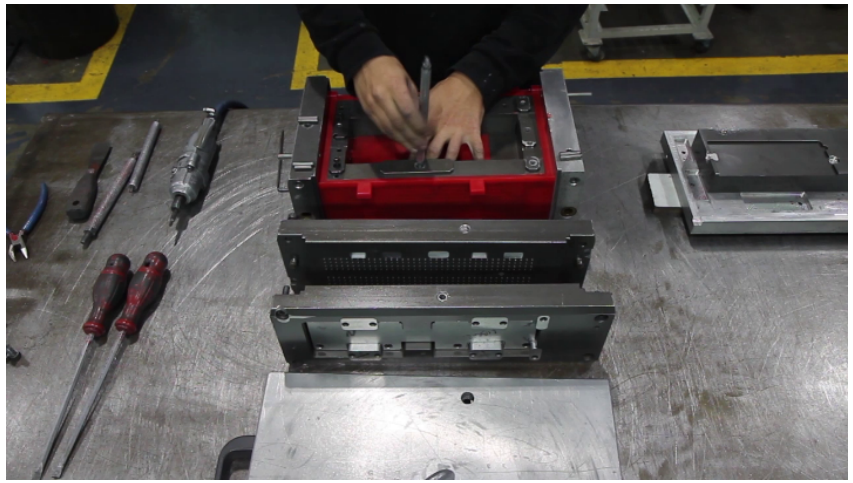
- Horizon 2020 research project
  - Call FoF-06-2014 “Symbiotic Human-Robot Collaboration for safe and dynamic multimodal manufacturing systems”
  - Coordinated by FUNDACION TEKNIKER (Spain)
  - http://fourbythree.eu/

Objectives of the Project

- Develop a new generation of **modular** industrial robotic solutions that are suitable for **efficient** task execution in collaboration with humans in a **safe** way and are **easy** to use and program
  - **Vision**: A system integrator (or end-user) can create its own custom robot according to their application needs (“kit” of hardware and software components)
- Real-world Pilot case studies to test Human-Robot Collaboration
  - **ALFA**, WOLL, STODT, PREMIUM

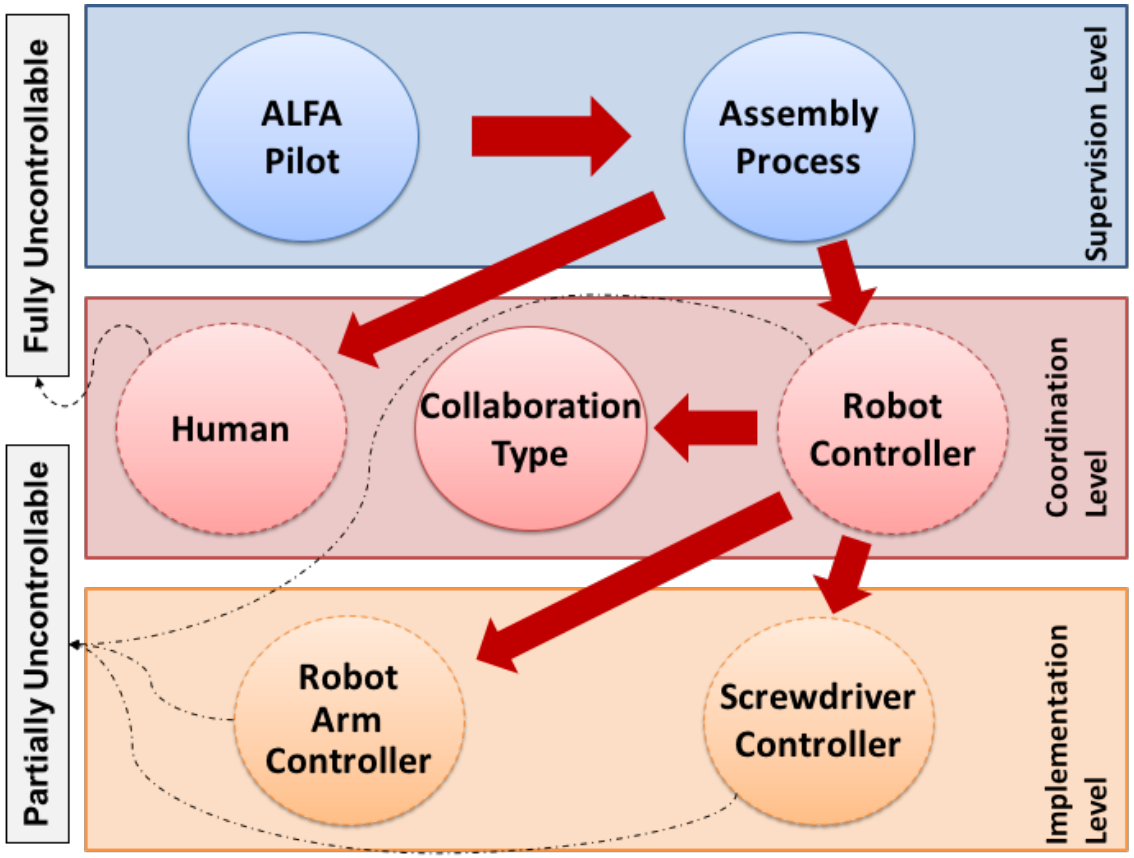
ALFA: A Collaborative Assembly Case Study

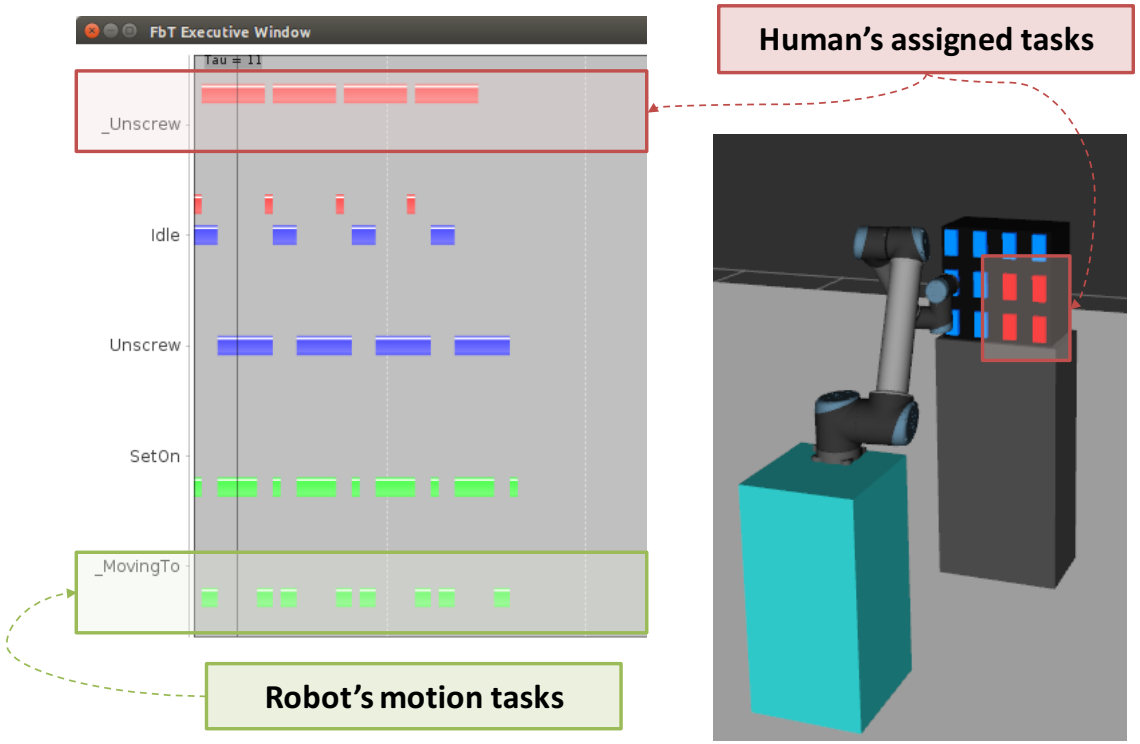




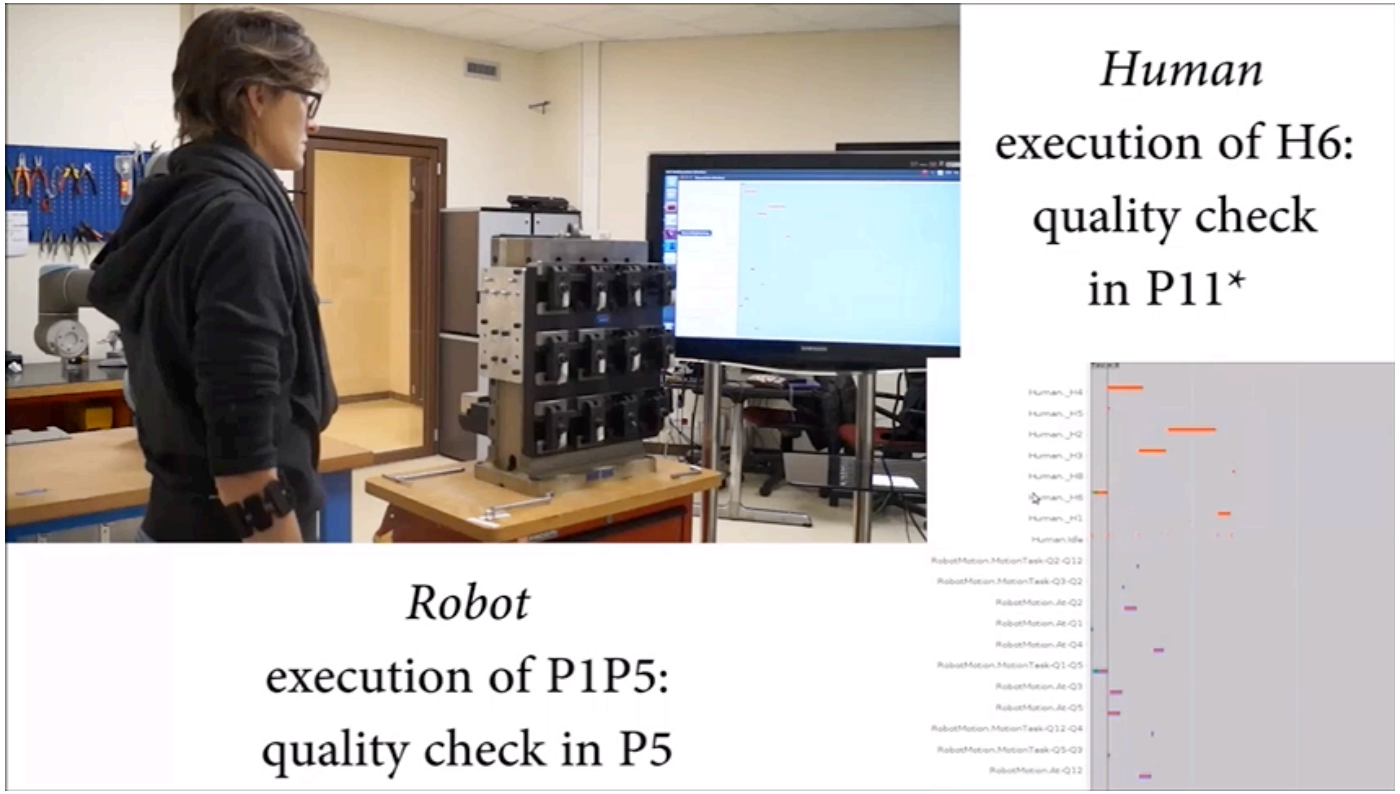
- Investment Casting process
  - Dies are assembled and disassembled manually
  - Some operations need human dexterity
  - Others can be done by a robot
- Re-design the process by taking into account an HRC perspective
  - **Hierarchical process description**
  - Three levels: **Supervision**, **Coordination**, **Implementation**

A Hierarchical Planning Model for Collaborative Assembly Scenarios



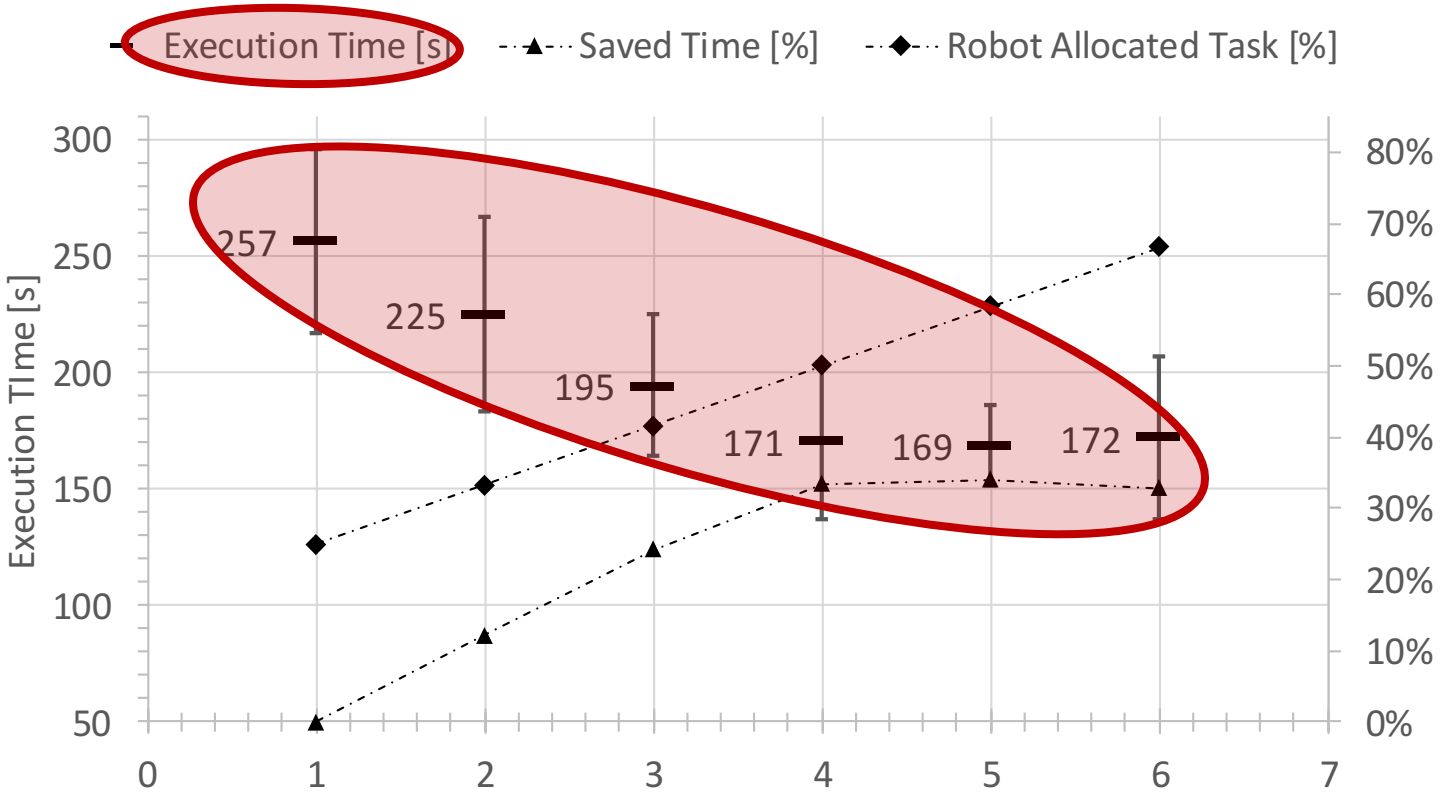


Flexible Collaboration with Dynamic Trajectory Selection



Verify the capabilities of an integrated task and motion planning control to improve the efficiency of HRC assembly processes

Experimental Results



Planning *can reduce* the duration of HRC processes and realize effective and safe collaborations between human operators and robots [Pellegrinelli et al. CIRP Annals 2017]

Final Remarks

- **HRC** is particularly suited to show the capabilities of PLATINUm
  - **Temporal uncertainty** and **hierarchical problem modeling and solving**

- The FourByThree research project has shown the capability of PLATINUm to **synthesize flexible and adaptable behaviors of a robot acting in the real-world**

- Download PLATINUm from GitHub: https://github.com/pstlab/PLATINUm

CORRELATED PROJECTS AND FUTURE DEVELOPMENTS

KEEN– The Knowledge Engineering Environment for Timeline-based Planning

- A knowledge engineering environment aimed at supporting the design and development of timeline-based applications
  - **Round-trip-engineering** to support both graphical and programming user interface to model timeline-based domains
  - Available as plugin for the Eclipse IDE
  - https://ugilio.github.io/keen/
- **Integrated with PLATINUm**
  - Launch platinum-based planners on designed timeline-based domains
  - Launch platinum-based executive to simulate execution of synthesized plans

Conclusions and Future Works

- PLATINUm is an open framework which can be “easily” extended to introduce new representation features and solving capabilities
  - Discrete and Reservoir Resource management has been recently introduced **[Umbrico et al. ICAPS 2018]**

- Future works
  - Investigate new heuristics and new search strategies
  - Integrate dynamic controllability checking algorithms
  - Comparison with other state of the art timeline-based and hybrid planners
    - EUROPA [Barreiro et al., 2012], CHIMP [Stock et al., 2015], FAPE [Dvorak et al., 2014]

Thanks for your Attention!

PLATINUm – **PL**anning and **A**cting with **TI**meli**N**es under **U**ncertainty

Consiglio Nazionale delle Ricerche

ISTC - CNR

Available on GitHub
https://github.com/pstlab/PLATINUm