Modelling Motion, Perception and World Model Stacks

Or, how to apply the RobMoSys models to hard realtime too?

Herman Bruyninckx, KU Leuven – TU Eindhoven

16 October 2018, MODELS 2018, Kopenhagen

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732410.

https://robmosys.eu

https://discourse.robmosys.eu

Examples of simple realtime “Activities”

All of these “Activities” can be done with 95% the **same modelling** patterns/policies/best practices as for “Components”!

“Realtime Activities”: often library/API-centred

“Port”-based over “API”-based

Ports *decouple*:

- **I** use a *port* when I need *data*
- **the system** architecture satisfies the *constraints* on *freshness of data* at every port

APIs couple:

- **I** call a *function* when I need *data*
- **I** am responsible for satisfying the *constraints*

Ports:

- *Port fan-in/fan-out hell*
  (**can** be overcome by *system architecture*)

APIs:

- *API explosion hell*
- *yet*, still most popular choice for *in-component* software...

Block-Port-Connector and Information Architectures

Block: hides functionalities

Port: view on part of the data processed in a Block's functionalities

Connector: models **ideal constraint** that data in connected Ports is **the same**

Information Architecture:

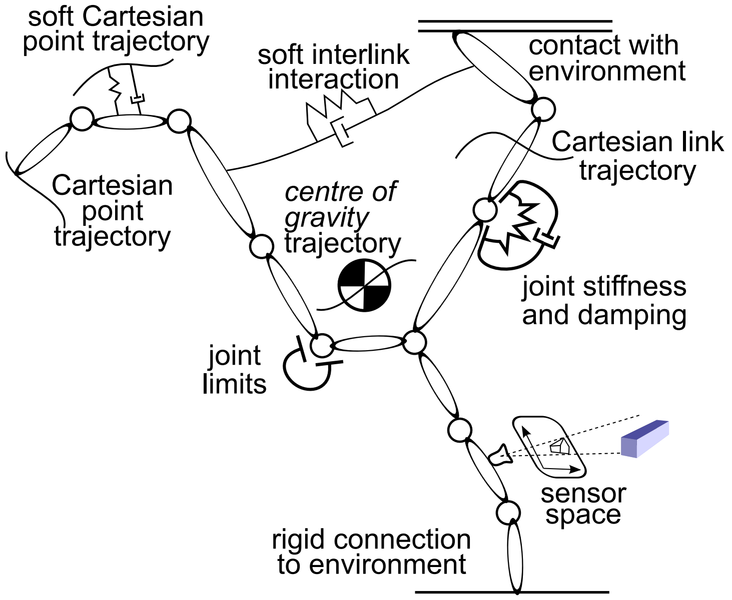
- major decision: what Ports and Connectors **not** to provide
- *the* place to model/apply *policies* on the **interaction dependencies** between Ports and their Connectors

Insight:

- the *same* models/patterns/best practices apply to “components” as well as to “functions”
- *also* (most) of the *policies*

Motion stack = coupling of control, perception and world modelling

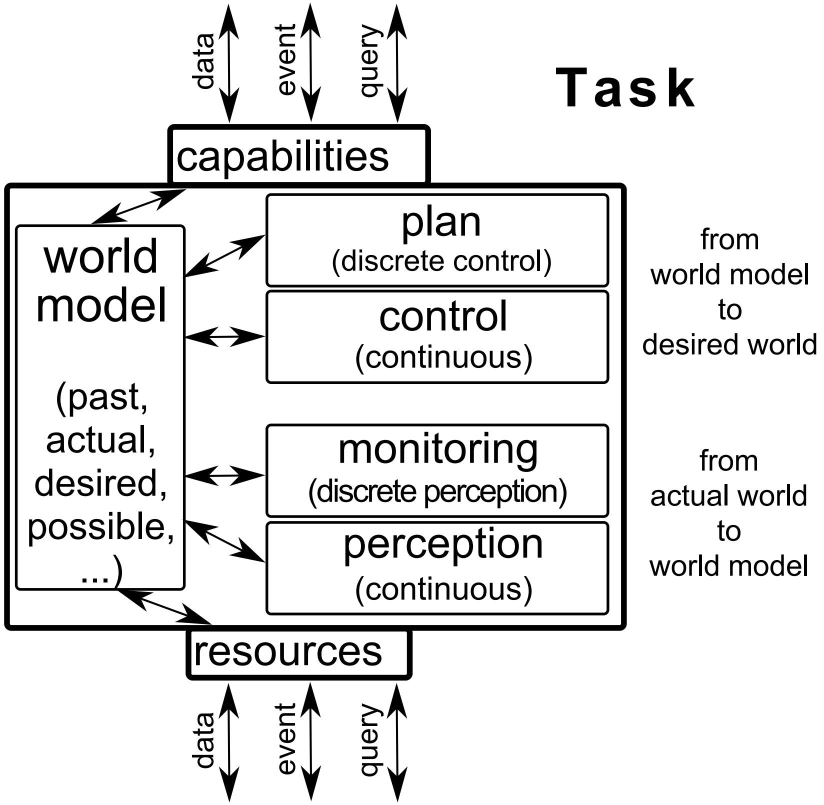
- Cartesian space, over *horizon*
- **Cartesian** space, *instantaneous*
- **Joint** space
- Transmission
- Actuator
- Battery/Power



This presentation focuses on only the joint space ↔ Cartesian space levels.

⇒ *robot* is largest part of *World Model*.

Motion stack – The meta model

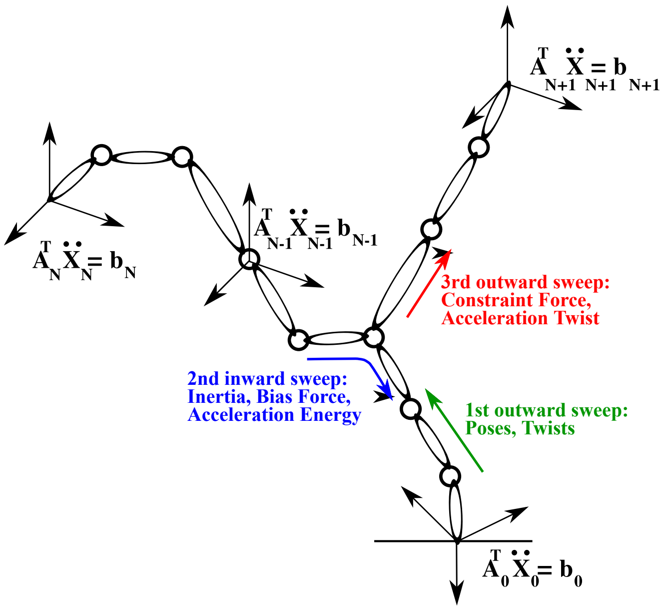


Information architecture: "arrows" = the *Ports* that must be *Connected*, with a *decision* of *Policy*

Insigths of information architect:

- **the** core kinematics & dynamics solver function
- which Ports **to couple**
- which Connector **dependencies** to configure

The core Kin&Dyn solver function



Three "sweeps" to produce/consume **all Ports**:

- *Outward 1*: positions, velocities, accelerations (**+** which *Ports* must be imported/exported!)
- *Inward*: forces, inertias, constraints
- *Outward 2*: joint torques

Physical state of kinematic chain: computed in *Outward 1* and *Inward*

→ **no policies** available: physics is what physics is!

Task state: computed in *Outward 2*:

→ **lots** of policies possible!

The core Kin&Dyn solver function (2)

begin

for $i \leftarrow 0$ **to** $N-1$ **do**

$$
\begin{aligned}
&{}^{p_{i+1}}_{p_i}\boldsymbol{T} = {}^{d_i}_{p_i}\boldsymbol{T}\;{}^{p_{i+1}}_{d_i}\boldsymbol{T}(q_i)\ ;\\
&\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i + \dot{q}_{i+1}\boldsymbol{Z}_{i+1}\ ;\\
&\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \boldsymbol{r}^{i+1,i} \times \boldsymbol{\omega}_i\ ;\\
&\boldsymbol{X}_{b,i+1} = \begin{pmatrix} \dot{q}_{i+1}\boldsymbol{\omega}_i \times \boldsymbol{Z}_{i+1} \\ \boldsymbol{\omega}_i \times (\boldsymbol{r}^{i+1,i} \times \boldsymbol{\omega}_i) \end{pmatrix}\ ;
\end{aligned}
$$

for $i \leftarrow (N-1)$ **to** 0 **do**

$$
\begin{aligned}
&\boldsymbol{P}_{i+1} = \boldsymbol{1} - \boldsymbol{M}_{i+1}(\boldsymbol{Z}^T_{i+1}\boldsymbol{M}^a_{i+1}\boldsymbol{Z}_{i+1})^{-1}\boldsymbol{Z}^T_{i+1}\ ;\\
&\boldsymbol{M}^a_i = \boldsymbol{M}_i + \boldsymbol{P}_{i+1}\boldsymbol{M}_{i+1}\ ;\\
&\boldsymbol{F}_i = \boldsymbol{P}_{i+1}\boldsymbol{F}_{i+1} - \boldsymbol{M}^a_{i+1}\boldsymbol{Z}_{i+1}(\boldsymbol{Z}^T_{i+1}\boldsymbol{M}^a_{i+1}\boldsymbol{Z}_{i+1})^{-1}\tau_{i+1} + \boldsymbol{F}^b_i + \boldsymbol{F}^e_i\ ;\\
&\boldsymbol{A}_i = \boldsymbol{P}_{i+1}\boldsymbol{A}_{i+1}\ ;\\
&\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i+1} + \boldsymbol{A}^T_{i+1}\left\{\ddot{\boldsymbol{X}}_{i+1} + \boldsymbol{Z}_i D^{-1}\left(\tau_{i+1} - \boldsymbol{Z}^T_i(\boldsymbol{F}_{i+1} + \boldsymbol{M}^a_{i+1}\ddot{\boldsymbol{X}}_{i+1})\right)\right\}\ ;\\
&\text{with}\quad D = \boldsymbol{Z}^T_i\boldsymbol{M}^a_{i+1}\boldsymbol{Z}_i, \quad \text{and} \quad \boldsymbol{\beta}_N = 0\ ;\\
&\boldsymbol{\mathcal{Z}}_i = \boldsymbol{\mathcal{Z}}_{i+1} - \boldsymbol{A}^T_{i+1}\boldsymbol{Z}_{i+1}D^{-1}_{i+1}\boldsymbol{Z}^T_{i+1}\boldsymbol{A}_{i+1}, \quad \boldsymbol{\mathcal{Z}}_N = 0\ ;
\end{aligned}
$$

$$
\boldsymbol{\mathcal{Z}}_0\boldsymbol{\nu} = \boldsymbol{b}_N - \boldsymbol{A}^T_0\ddot{\boldsymbol{X}}_0 - \boldsymbol{\beta}_0\ ;
$$

for $i \leftarrow 1$ **to** N **do**

$$
\begin{aligned}
&\ddot{q}_i = (\boldsymbol{Z}^T_{i-1}\boldsymbol{M}^a_i\boldsymbol{Z}_{i-1})^{-1}\left\{\tau_i - \boldsymbol{Z}^T_{i-1}\left(\boldsymbol{F}_i + \boldsymbol{M}^a_i\ddot{\boldsymbol{X}}_{i-1} + \boldsymbol{A}_i\boldsymbol{\nu}\right)\right\}\ ;\\
&\ddot{\boldsymbol{X}}_i = \ddot{\boldsymbol{X}}_{i-1} + \ddot{q}_i\boldsymbol{Z}_i + \ddot{\boldsymbol{X}}_{b,i};
\end{aligned}
$$

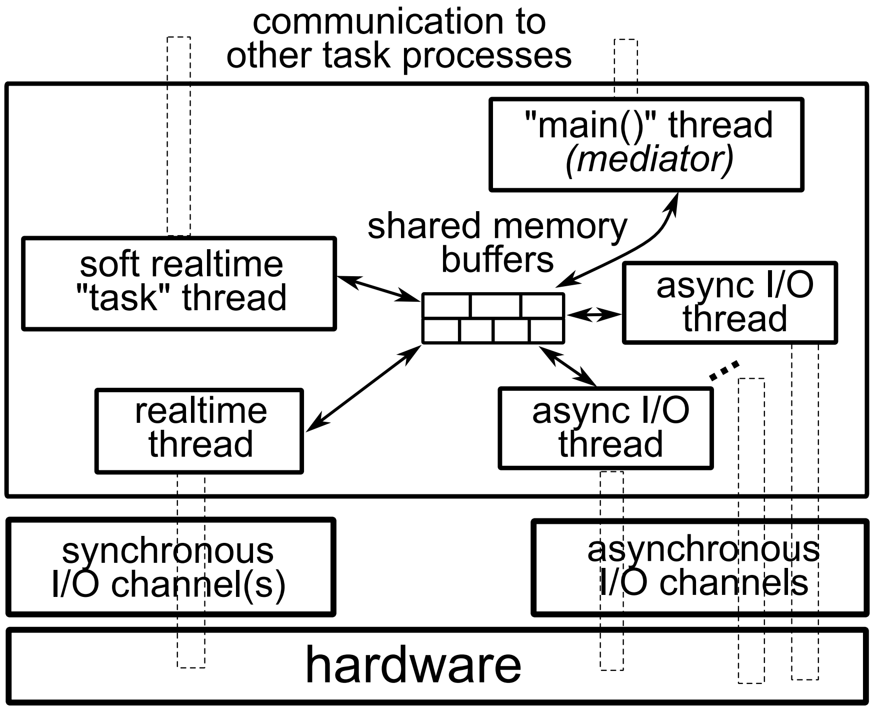
Third "sweep" policies:

- saturation detection
- output scaling
- priorities or weighing
- joint limits
- Model Predictive Control
- estimation & identification
- compensate friction/elasticity
- ...

Note: no *Jacobians* needed!

From Information Architecture to Software Architecture

1. (Real-time) process pattern



Policies:

- buffering: what? communication patterns? garbage collection?...
- **wait-free streams**!? (*not available* for components!)
- first/second "sweeps" in hard or soft realtime
- add monitoring/ heartbeat
- ...

This *Activity* can (but need not) be a *Component*.

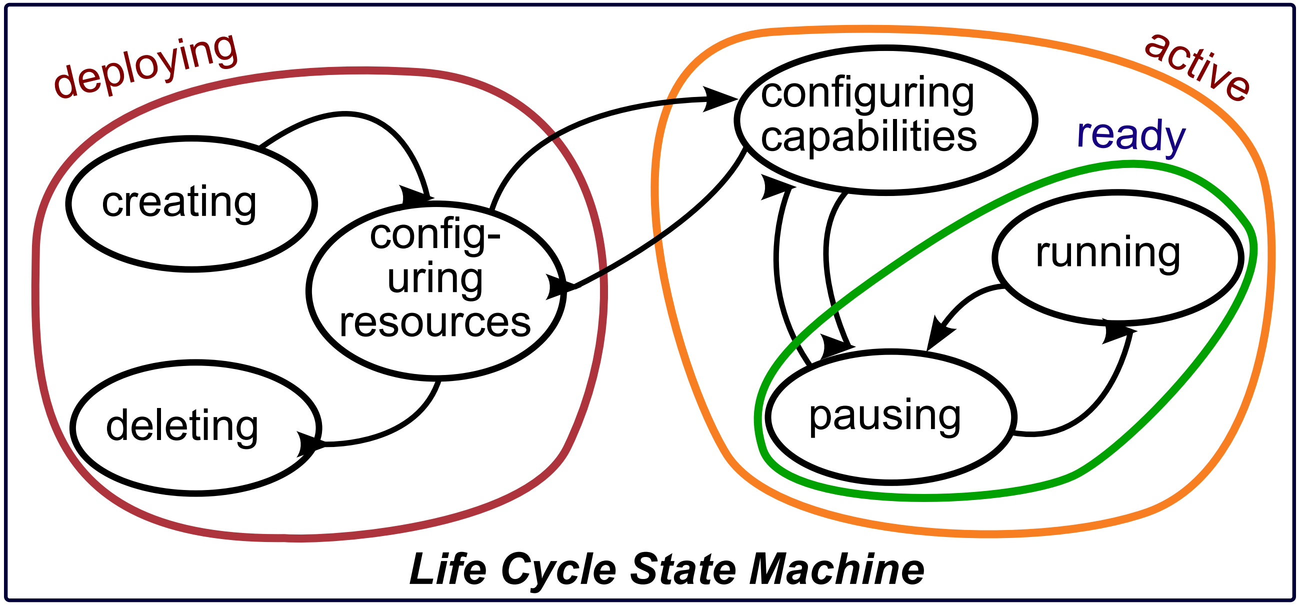
2. Event loop(s)

```
when triggered // by operating system, which deals with all asynchronous side effects.
do {           // the control flow structure of the event loop.
communicate()  // get all "messages" with events & data, filled in by other asynchronous activities.
coordinate()   // handle the events in these messages, and decide which ones to react to.
configure()    // some events imply reconfiguration of computations.
compute()      // execute your (serialized set of) synchronous algorithms,
               // which in themselves are side effect-free computations.
coordinate()   // the computations above can generate events that
               // imply reconfiguration of this event loop.
communicate()  // the computations above can generate events & data that
               // other asynchronous activities must know about.
sleep()        // until the shortest deadline
   }
```

The loop's **schedule(s)** are **only difference** with component approach

but, they can use the **same dependency model**

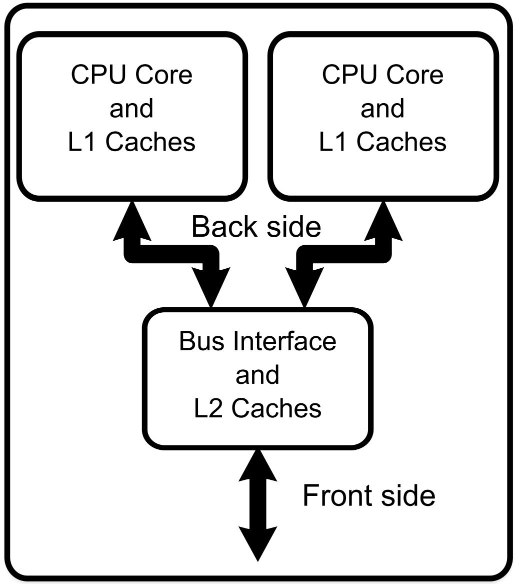
3. Life Cycle State Machine(s)



One LCSM per **activity**. Including the Activity **to coordinate** Activities.

More than one LCSM possible per event loop

Deployment on multi-core CPU



Similar information architecture of Block-Port-Connector compositions between data storage, data fetching, and CPU computations.

→ **new** policies *may* have to be configured (introduced...) for this level of deployment.

Very similar considerations apply when deploying to **physical communication** infrastructure, e.g., EtherCat, Sercos, UART,...

Medium-term RobMoSys results

- **models** for realtime motion stack
  → *drafts* for standardization
- **models** to **configure** which “solver” *Ports* one needs at **component** level
- compositions with realtime *world model* and *perception*
- reference implementations
- deployment via *modelling* of *configuration* of the reference implementation

(**You** can contribute to this, via a RobMoSys *ITP Grant*!)

Thank you for your attention.