Toward Self-Adaptive Software Employing Model Predictive Control

NII Shonan Meeting on Controlled Adaptation of Self-Adaptive Systems (CASaS)
Shonan, Japan, April 24-28, 2016

Holger Giese, Thomas Vogel, and Sona Ghahremani
Hasso Plattner Institute
University of Potsdam, Germany
holger.giese@hpi.de, thomas.vogel@hpi.de
http://hpi.de/giese/
What is Model-Predictive Control?

“Model predictive control has had a major impact on industrial practice, with thousands of applications world-wide.”

[Seborg+2011]

Idea of Model-Predictive Control (MPC):

- Make required control decision based on predictions for a model of the controlled process by solving a related optimization problem (e.g., maximizing a profit function, minimizing a cost function, maximizing a production rate) at runtime.
- Usually MPC is running on top of simpler controllers (e.g., PID) that control the subsystems of the process according to the control inputs from MPC (hierarchical control).

Capabilities:

- Can handle complex MIMO processes
- Can realize different optimization goals
- Can handle constraints on the control inputs and process outputs/state
- Can compensate loss of actuators (determine control structure + check for ill-conditioning)
- Can be combined with online identification

Remark: also named moving horizon control or receding horizon control
Advanced MPC in Terms of MAPE-K
Mapping Advanced MPC to classical MPC

Advanced MPC:

Classical linear MPC:

[Seborg+2011]
Finite Receding Horizons in MPC

- (prediction horizon – control horizon) * sampling time ≈ settling time (horizons = number of considered steps)
- Sequence decision problem (agents)
Example: Self-Repair

- Failures of different types:
  - Various exceptions
  - Crash of a component
  - ...

- Multiple repair strategies for each failure type:
  - Restart the component
  - Redeploy the component
  - Replace the component
  - ...

1. Which strategy should be applied to repair a specific failure?
2. If there are multiple failures, which one should be repaired first?
Example: MAPE-K with EUREMA & MORISIA

EUREMA
(Executable Runtime Megamodels)
mdelab.de/mdelab-projects/software-engineering-for-self-adaptive-systems/eurema/

MORISIA
(Models at Runtime for Self-Adaptive Software)
mdelab.de/mdelab-projects/software-engineering-for-self-adaptive-systems/morisia/

Adaptation engine

Analysis rules

Plan rules

Failure meta model

Performance meta model

Architectural meta model

EJB meta model
Example: Analysis & Plan - Which strategy to apply?

- Predicting two steps, **Restart** appears to be the better strategy.
- Predicting seven steps, **Redeploy** appears to be better (e.g., using a different node with more resources).
- Short vs. long term (steady state **utility** dominates **reward**).
Example: Analysis & Plan – Which failure to repair first?

Explore the strategies for the different failures (f1 and f2):
• Steady state utility is the same but order matters considering the reward
• Repair the failure first whose repairing improves most the reward (f1)
Utility-Based View of the Solution Space

- **Analysis**: Check whether the current state is optimal concerning its utility
  - **Static optimization**: Check whether a better optimal solution state exists. (side-effect is that we also have one optimal/satisficing goal state)

- **Planning**: Find a path with optimal reward leading to the chosen solution
  - **Dynamic optimization**: what is the optimal path to the chosen solution state
  - Trivial in case solution space can be easily configured
Cases for the Selection of the Horizons

• Solution space is not fragmented (you can compensate “failures” ...)  
  → (small) finite horizon may be sufficient
• No or unlikely interference with process behavior  
  → usually 0 settling time  → prediction horizon = control horizon
• Multiple control inputs feasible in one control step  
  → receding horizon may be skipped or “reduced”
Beyond Classical and Advanced MPC

- **Infinite horizon** can lead to better results (if long term predictions are accurate), as it considered the steady state assuming optimal behavior, but it requires more resources.

- **Stochastic MPC** considers probabilities for process behavior and optimizes the *expected reward*.

Beyond advanced MPC:

- For **non-deterministic models** (e.g. PTA) the control inputs (strategy) requires to be safe (any or too high risk is avoided by excluding unsafe control options).

- Agents **learning the expected rewards** (not via state) leads to predict reward rather than process behavior.
Beyond MPC: Layered Architecture & Adapt

- Adapt MPC (monitor, analysis, plan, execute)? e.g., adapt rules, attention
- Adapt underlying controllers (omitted in the architecture)
Conclusions & Outlook

• MPC can handle many properties of complex process models typically present for software (MIMO, different optimization goals, constraints on the control inputs and process outputs/state, loss of actuators)
• Advanced MPC seems suitable as a framework to understand and fine-tune many approaches based on models and related predictions.
  – Can employ for a variety of techniques (simulation, optimization, search, synthesis, ...) and models (linear, non-linear, state space, probabilistic) ...
• The horizons for control and predictions result in a useful design space in many cases (depending on the characteristics of the state space).
  – Enlarging the control and prediction horizon can help to engineer more accurate solutions (infinite = optimal?)
  – Limitation of the control and prediction horizon (and also input blocking) can help to engineer better scalable solutions
• But: MPC with bad models of the process don’t work!

