Towards Software Model Checking in the Context of Model-Driven Engineering

**Motivation**
- Software models help structuring large software projects
- Fixing software design errors late is expensive
- Verification techniques for software models necessary
- Hardware models are currently verified by model checking
- Conceptual gap between software and hardware models prevents widespread adoption of model checking for software models
- Problem: How can we close this gap?

**Our approach**
- **cOCL**
  - Two OCL extensions with formally defined syntax and semantics
  - A Computational Tree Logic (CTL) extension for verification
  - A selector extension for querying interesting system states
- **MoCOCL**
  - Implementation including visual result inspection
  - State space generation using the graph transformation tool Henshin
  - Extending the XText OCL Engine
  - State space and model visualisation and traversal using web technologies

**Read more about MoCOCL...**
- Demonstrated in terms of Pacman
  - Metamodel (Ecore)
  - Initial state (XMI)
  - Model behavior (Henshin rules)
  - Is this implementation correct?
  - System verification
    - No more turns are possible if Pacman has found the treasure
      - **Always Globally** pacman.on.treasure implies (Always Next false)
    - MoCOCL builds the statespace and calculates the relevant parts

**MoCOCL tool output**
- Interactive traversal of calculated cause
- Includes only relevant state space parts and highlights relevant model parts for easier debugging

**Further problem diagnostic**
- Select all offending states
  - **self@**
    - where pacman.on.treasure and (Exists Next true))
  - Is a ghost moving in all offending states?
    - **Always Globally** pacman.on.treasure implies \( (\text{Always Next false}) \implies \text{ghosts} \rightarrow \text{collect}(g \mid \text{let} o = \text{on in self\{next having g.on} < o\}) \rightarrow \text{includesAll}(\text{self\{next\})}
    - Yes, so the ghost move rule might be wrong

**Performance evaluation**
- Scalability of various Pacman fields
  - Small (S)
  - Medium (M)
  - Large (L0, L1, L2) with 0 to 2 ghosts

**Usability evaluation**
- Evaluated the CTL extension with 11 participants
- Participants had to solve tasks and give their opinion
- Subjective Evaluation results
  - Reading cOCL: Easy
  - Writing cOCL: Medium
  - Using the tool: Easy