Position paper:
discussion during and after the presentation
are very welcome!

Mika Siikarla
mika.siikarla@tut.fi

Tampere University of Technology (TUT)
Department of Software Systems
Outline

- MDE & TDE (+SST)
- Position: TDE is bad for evolution; embrace interaction
- How TDE hinders evolution
- Why SSTs are not realistic
- What shortcuts are made with SSTs
- Why SSTs are not a good ideal goal
- Model marking vs. interactive model transformation
MDE, TDE, SST

- **MDE**
  - For general use in SW development (like OOD)
  - Focus on models; MTs keep models in sync
  - Make design decisions explicit in models
  - Decide and design languages for information (models)
  - Design phases better defined and possibly split into smaller steps
  - Assumed to help evolution

- **TDE**
  - Focus on transformations; models help MTs operate
(MDE), TDE, SST

- **TDE goals**
  - MTs can be linked to form an automatic chain
  - Only m1 needs to be maintained
  - SW product can be fully specified indep. of tech
  - HW and SW platforms and details cheaply changed
  - Product developers update m1 and select/build MTs

- **Heavy quality requirements on MTs and must be cheap**
  => Shiny Silver Transformation (SST)
  - Complete and correct
  - Fully automated
  - Already exists or easy and fast to develop
  - Does not need to be maintained

- **SSTs not usually explicit goal**
- **SSTs not claimed; still TDE ideals**
Position

- SSTs not realistic or practical in general case
  - None of the features can be reached
- Not practical => TDE goals should not be expected
- Aiming for SST is bad
  - High quality reqs => heavy implementation
  - Implementing guesses wastes effort
  - Evolvable mass increases; heavy adds more mass
  - Guesses give false confidence + hide weaknesses
- Aim to assist designer instead
  - (start with) interactive transformations
  - Model decisions, don't remove them (guide)
  - Don't implement wild guesses, let user sort them out
  - Automate (only) if possible and sure (or not)
MDE and TDE work flows

one design phase

req → design decisions

\[ m_1 \]

\[ m_2 \]

\[ m_3 \]

impl

design decisions

design decisions

design decisions

impl

chain of transformations
How SST hinders evolution

- High quality reqs => very demanding to develop
- Become part of build environment and must be maint.
- Lock-in or at least much effort
- Heavy development may pressure into bad reuse
  - Restrictions come from tool, not use!
- The earlier tech. decisions are made, the more wrong
- Design phases are defined by information not form
  - Modeling language may change
  - Grutchko et al. typically change
- Option; light-weight MTs
  - Interaction => flexibility + less implementation
  - Only implement what is known => less bad guesses
  - Not as restricting and lighter to change
SST not realistic: Exists already?

- Only if the problem or a more general one solved
- To assume always exists is to assume all solved
- Similar project exists: MT more likely
  - Application domain, modeling languages, development process, company policies, technologies, platforms, protocols, ...
- No similar projects: MT less likely
  - Any unique or pioneer project
  - New concepts in SW frequent
  - New tools and technologies very frequent
- Only helps with the easier cases, the hard ones devel.
- Framework parallel
- Component, code reuse didn't work, MT reuse will?
SST not realistic: Easy and fast?

- If effortless, must be easier than SW devel in general
  - Hardly! Must solve a larger problem
- If needs to be devel, it is likely new and uncharted
  - Not just implementing complete precise specs
- Some SW projects already finished, so some info
  - No full understanding of all requirements
  - Just like any SW project w/ use cases
  - Understanding grows with more cases and devel
- Domain experts can not explain rules
  - May give examples
- Classical example: DB design
- Implementing not trivial either
  - optimizing compiler: man-decades or man-centuries
SST not realistic: Complete & correct?

- Heavy requirement
- All requirements are not known in the beginning
  - Would require psychic guesses
- M1->M2 is one big issue
  - More fundamental one is information in models
- Often target models must also be “good”
  - Optimization rules often more vague and complex
  - Sometimes more global: can't index all tables
- Necessary requirement: m1 contains all information
  - There are no design decisions!
  - Just a translation
- Concessions mean MT is not automatic => no chain => no TDE benefits
SST not realistic: Fully automatic?

- Saves effort, reduces human error, requires no skills, no operator
- Nice, but requires m1 contains all information
- No one really claims this
- Augment source model prior or fix target model after
  - Not automatic; or, not m1->m2, but m'1->m2
  - Design decisions, just in diff form
  - Can't form a chain
  - Can't get TDE
"The lifetime of a good compiler platform is typically in the 10-year (and above) range. (...) Finally, compilers are large programs. Over their lifetime, they become inflexible and difficult to modify, and they accumulate warts and scars that are frustrating to work around. Almost every component will be rewritten from scratch over the lifetime of the compiler, and more than just components may have to be replaced to accommodate new features." [7, p. 288]
What shortcuts are made with SSTs

- Model marking
  - Stereotypes in UML
  - MT uses markings to disambiguate between actions
  - Parts not understood/implemented left for designer

- Target model fixing
  - Problematic with decisions that cause large structural differences
  - E.g. all in one table vs. each in its own table

- Fully manual
  - Traceability is lost
  - No guidance
What shortcuts are made with...
What shortcuts are made with SSTs

- TDE goals can not be achieved
  - Not an automatic chain
  - Design decisions included in each phase
  - Not fully technology independently specified
  - HW/SW details can not be changed (new MTs)
  - Product developers need to do changes to all models and someone needs to develop the MTs
Why SST is not a good goal

- Keep the standards high?
- Hides the weaknesses + does not guide user
- Better: admit weaknesses and try to manage them
- If the designer must be involved, the focus should be in making his work easier and guiding him for good quality outcome
  - View MT as a tool for designer, not instead of
  - Try to model and manage interaction to guide user
- Build only parts that are known; leave rest to user
- Make user aware of limitations, don't hide them
- Start fast and start lean; reduce interaction later
- Degrade gracefully; solve sub-problems if possible
- Some interaction can be left in to keep simple
- (Interaction does not mean re-asking questions)
Model marking vs. interactive MT

- **MM**
  - All potentially needed information must be provided (in theory)
  - Marking is fully manual; no guidance
    - No traceability
  - Only know decision result
  - Can be done iteratively, but that's just bad IMT

- **IMT**
  - Provide only what is needed (“lazy evaluation”)
  - User can be guided (if dec. modeled)
  - Some traceability across decisions (if modeled)
  - Have information about the state of the MT
    - Helps with smart decision reuse
Summary

- MDE not a conveyor belt of transformations
  - Attempt to store design time rationale for later use
- Some automation to help model synchronization
- Transformation-assisted design, not automation
- Start lean and automate when sure and is possible
- Keep light-weight, possibly even when could automate
- Do not hide decisions, expose and try to manage