ECBS2004—Brno

Berkeley
University of California

Improving CBS Tool Development With Technological Spaces

Jonathan Sprinkle

University of California, Berkeley

Let's see, where to begin...

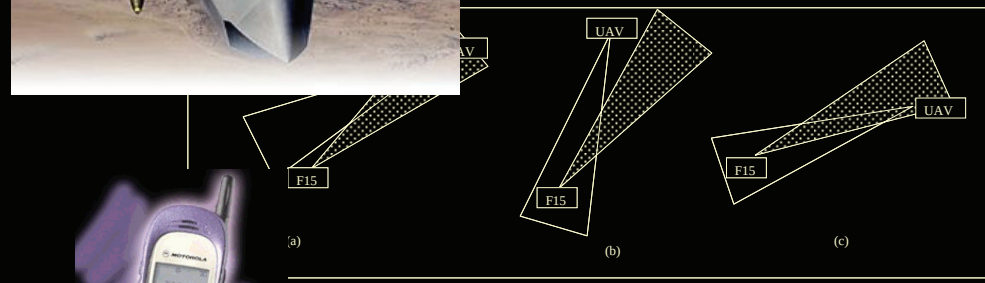
Begin at the ending

- People who ask me what I do, now come away with a better understanding (based on questions they ask)
- I have a way to link *transformations* with *output* formats, that accounts for semantics, as well as buzzwords
- Now, I have a way to help students' comprehension of what it is they will be doing in my class

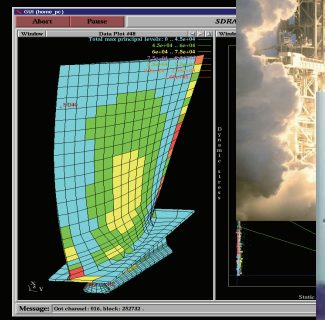
These computers these days…

- CBSs are used for many kinds of purposes and products
  - Fault diagnostics
  - Evasion strategies
  - Safety verification
  - Logistics planning
  - Mobile phones
  - Embedded systems
- Tricky part is, the software that runs the computers is nontrivial to produce








I remember when I was a kid…

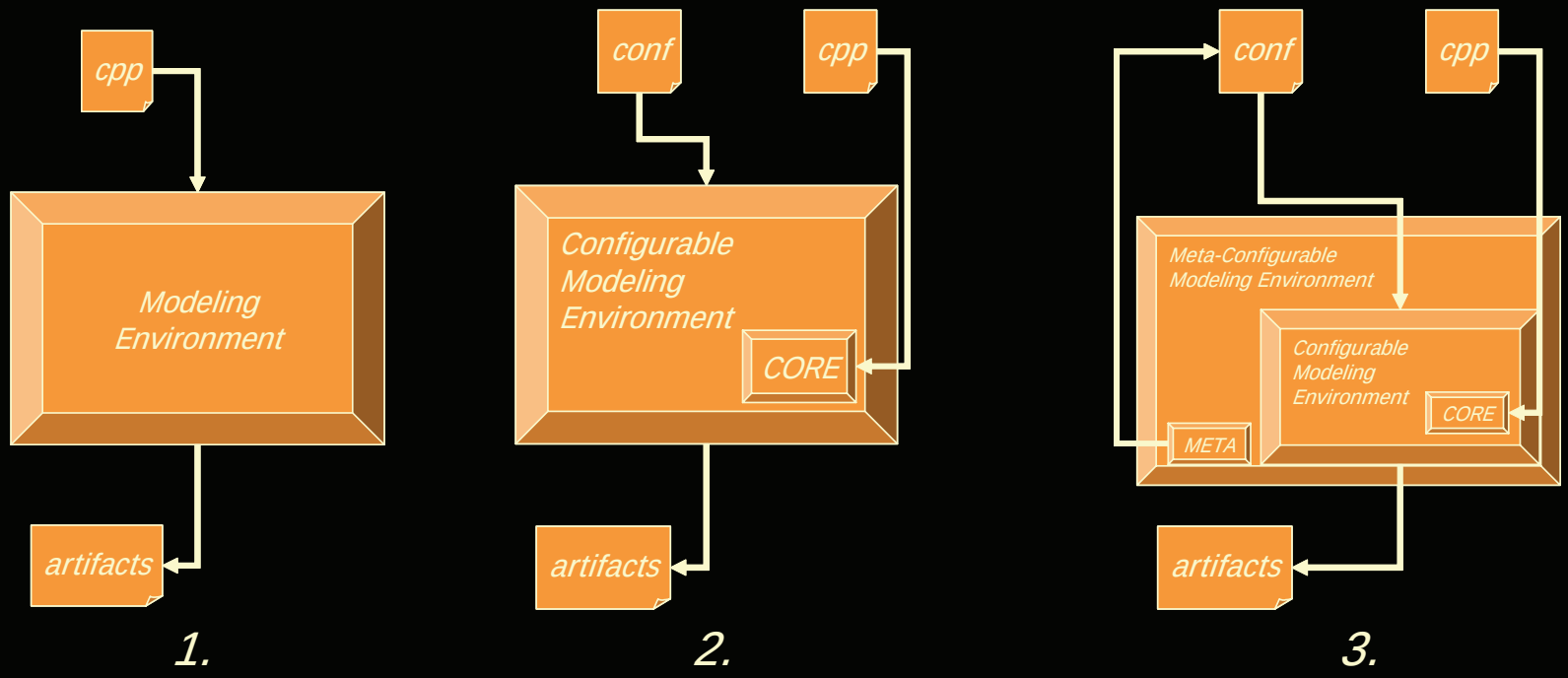
Berkeley
University of California

- Managing system level code from the code itself is an intractable problem
  - Too many crosscutting concerns
  - Tight coupling between language and semantic encoding
  - Small changes to requirements mean large changes to implementation

"Why, back when I was a budding programmer, we didn't even have keyboards!! And we worked our little hairy fingers to the bone. AND WE LIKED IT!"

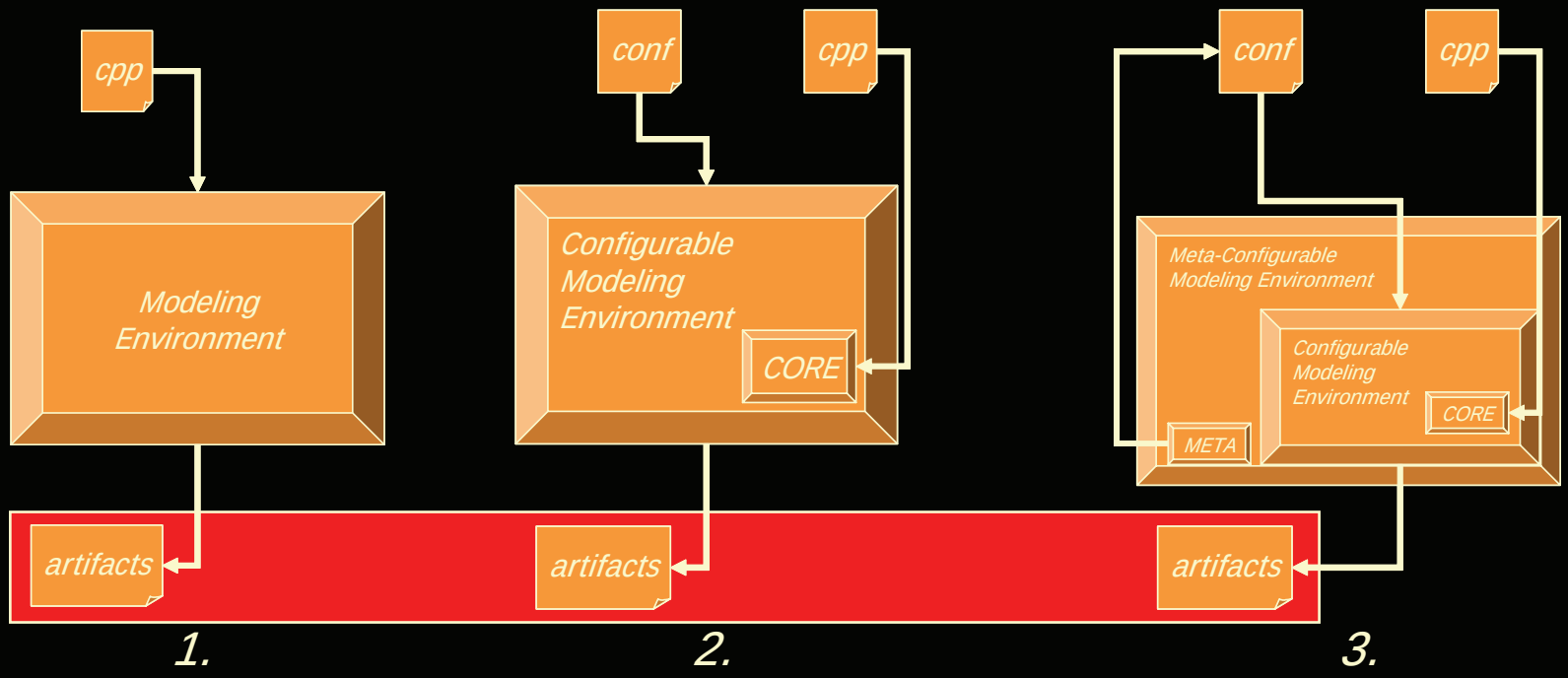
Enter DSMEs

- Domain-Specific Modeling Environments
  - First hard coded (hard to create and accurately encode)
  - Then configurable (abstracted visualization away)
  - Then meta-configurable (model the configuration)



But…

- There is still one piece that is difficult to describe
- Not just *formally* but *informally* as well



Berkeley
University of California

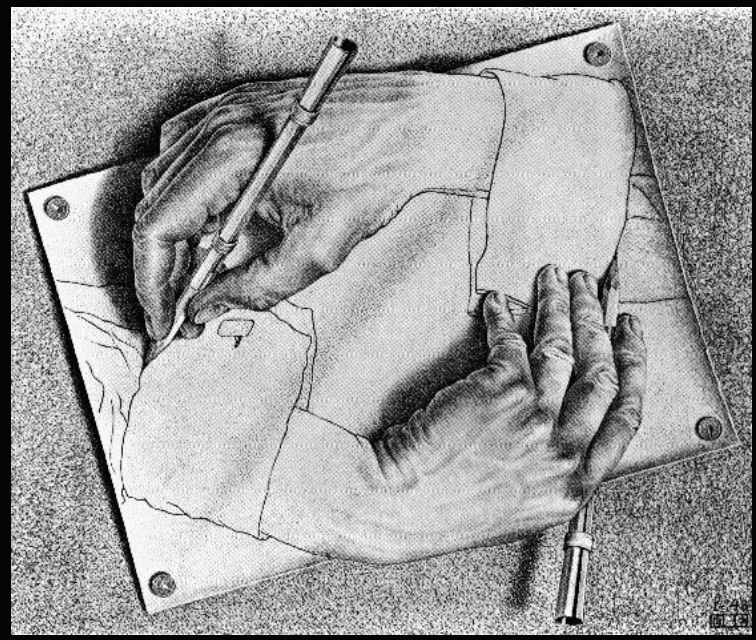
The question is…

- What are *artifacts*
  - The "outputs" of the tool
  - Text files, XML files, header files, C++, Java, Matlab…
- How can you *describe* artifacts without boring the pants off whomever you are describing them to?

ways said that a CBS tool that doesn't produce system configurations or generated code is like playing tennis without a net. Well, I could go on for hours, but I'd probably start to bore you…"

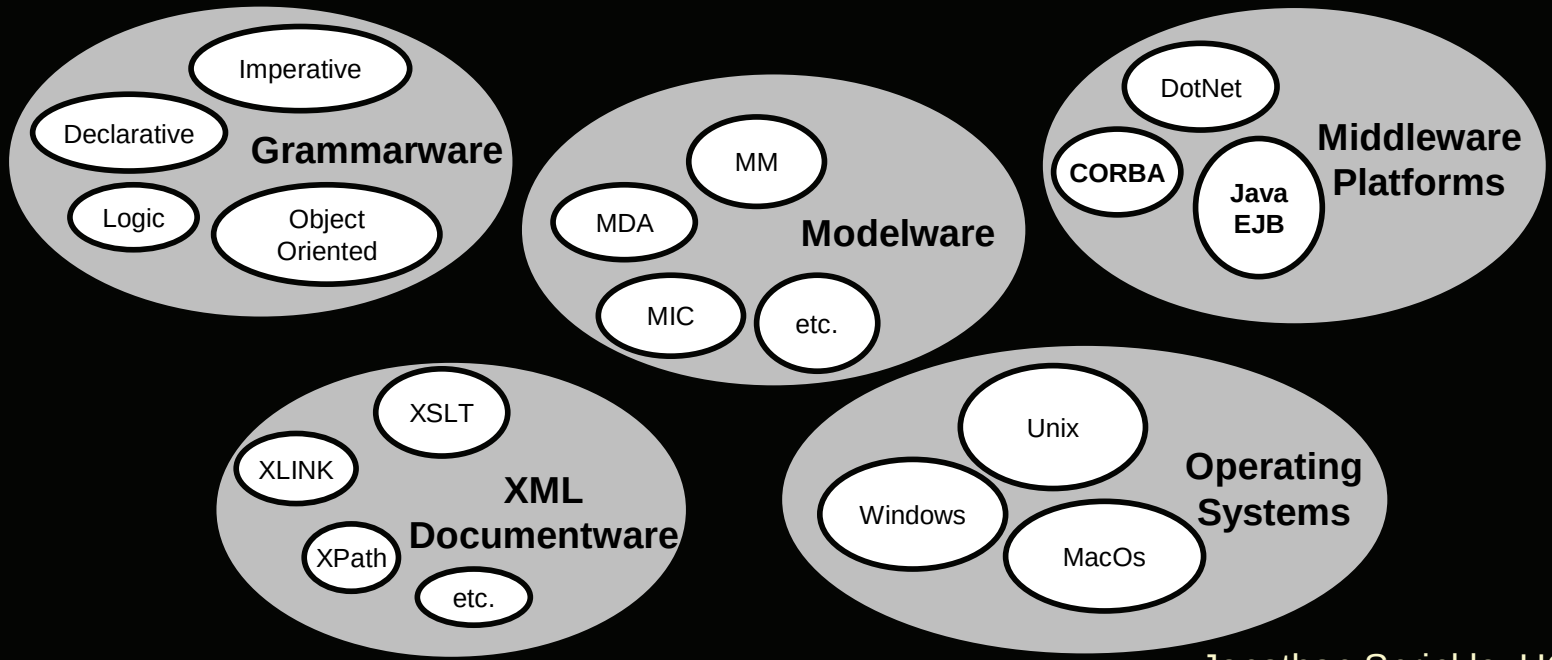
The answer is…

- A Technological Space (TS)
  - Context for expression of data, concepts, or execution space
- Members of a TS have some (or all) of the following in common
  - Metamodel
  - Syntax
  - Semantics
  - Domain
  - User base



Example TSs

- Operating systems
  - Unix, Linux, DOS, Windows…
- Middleware Platforms
  - CORBA, EJB
- XML
  - XML DTDs, XSL, XMI
- Modeling
  - MDA, MIC



So, how do you use them?

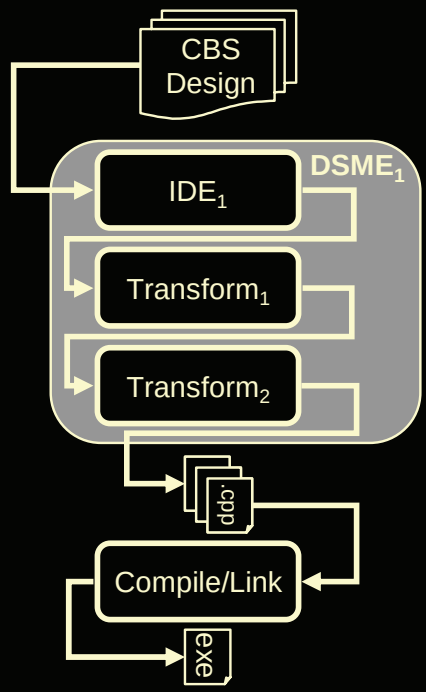
- Design considerations
  - Optimizing data flow through a transformation chain
  - Deciding 3rd Party tools to use
- Tool descriptors
  - An additional description of not *what* something is, but *how* that something is expressed.

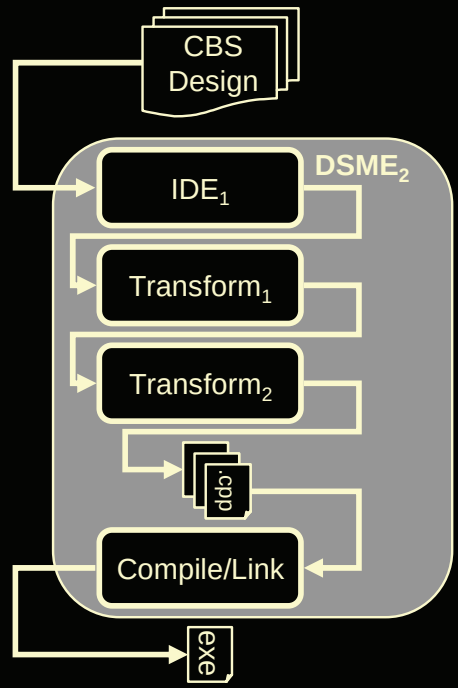
Ahh, Magritte's pipe. The complex expression of realism combined with the birth of abstract art and the (arguable) founding of post-modernist expression through paradox.

Or, a photograph in the JPEG TS of a painting in the oil-on-canvas TS

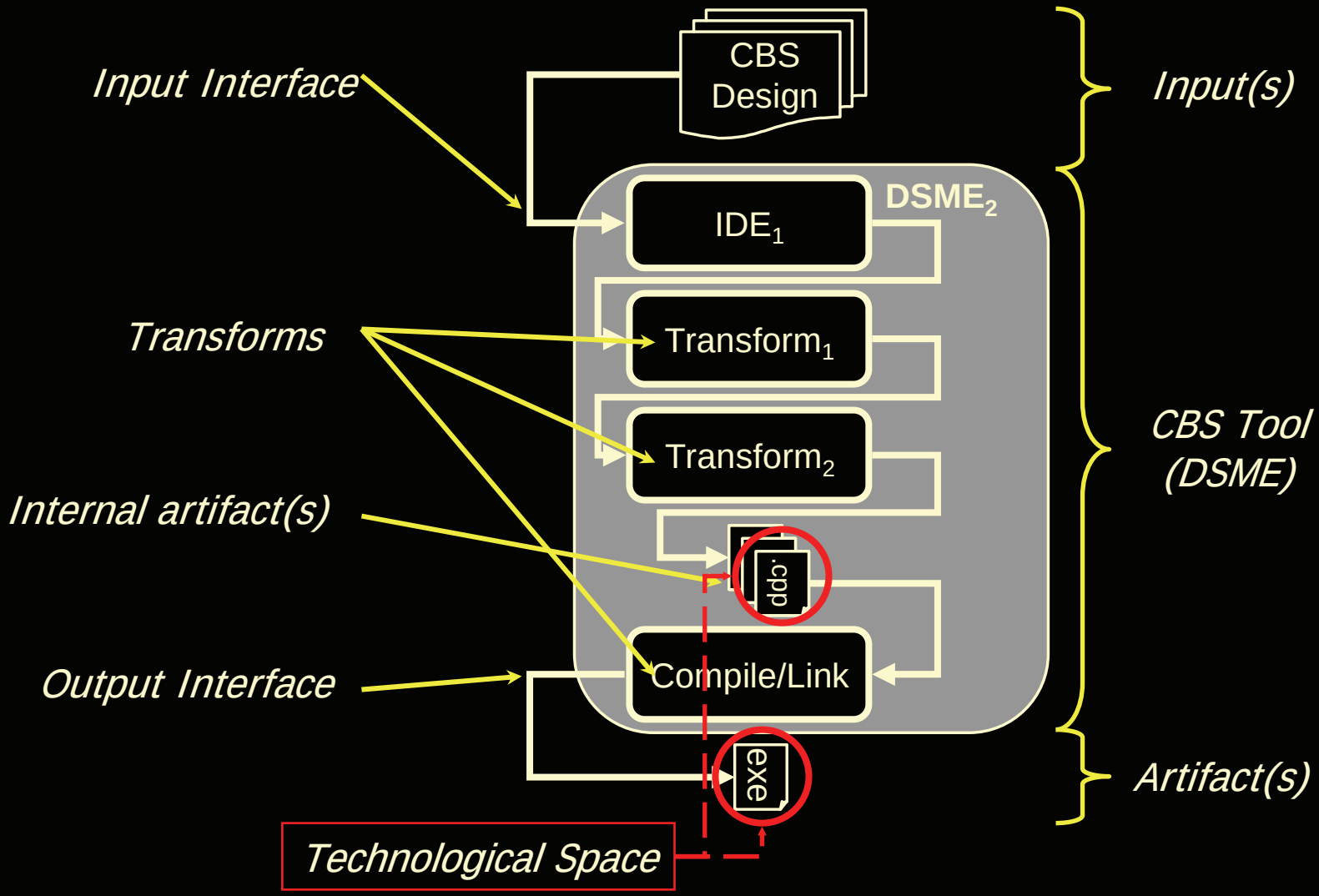
(Abstract) Example

- Two similar DSMEs (1 & 2)
- DSME1 produces
  - code in the *C++* TS, a.k.a.
  - text in the ASCII TS
- DSME2 produces
  - code in the *executable* TS, a.k.a.
  - binary information for a particular OS
  - still produces *C++* TS in an intermediate step
- Absorbing transformations to be part of the tool is called *absorption* (surprise!)





Using the TS formalism

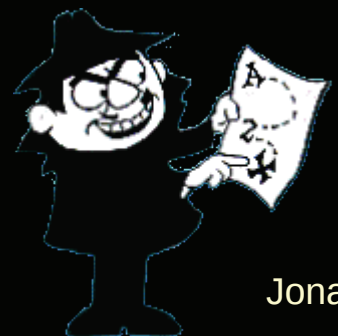


How does it help?

Inspiration!

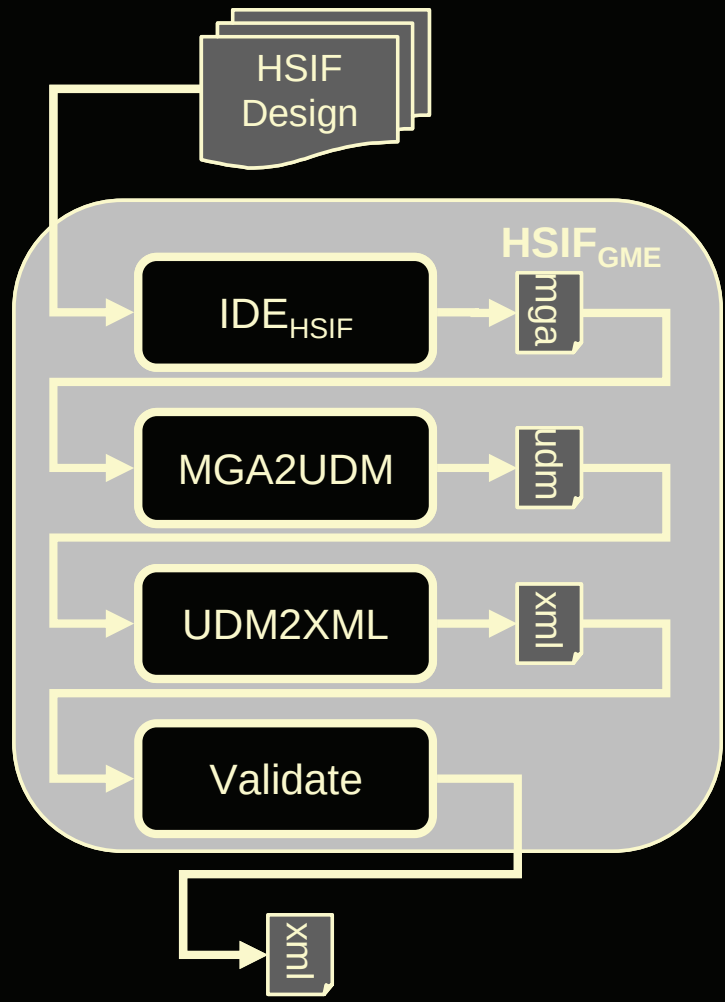
- Gives a high-level overview
  - (similar to a UML use-case)
  - Not used for code generation
  - Documentation, rather than design formalization
- Describes the flow of data through the DSME
- Explains the I/O role of intermediate artifacts
- Clarifies the role of 3rd Party tools in the chain of transformers/interpreters

"Don't worry, Fearless Leader, they have no plans for system integration! Mua, ha ha ha ha ha!"
-- Boris Badenov



(Concrete) Example

- Hybrid Systems Interchange Format (HSIF)
- Defined as an XML schema (.xsd) with accompanying semantics
- A DSME for HSIF modeling and file generation is termed $HSIF_{GME}$
- Performs several transformations on a set of input models to produce a final artifact useful to other hybrid systems tools



So what?

- Why not just create the XML file straight out?
  - Not easy to visualize the system
  - Difficulty in expressing equations in the XML format
- Thus, the $HSIF_{GME}$ IDE_{HSIF} differs from the artifact in syntax, to be more convenient for entry

$$\dot{x}_3 = x_1(1 - x_2/2)$$

```
- <DiffEquation>
 - <AExpr> <MExpr>
  - <MExprR mulOp="*">
   - <MExpr> <ParExpr unOp="NOP">
    - <Expr> <LExpr> <RExpr> <AExpr>
     - <AExprR addOp="-">
      - <AExpr> <MExpr> <MExprR mulOp="/">
       - <MExpr>
          <Const unOp="NOP" value="2" />
         </MExpr>
        </MExprR>
       <VarRef _id="id1d" var="id3" unOp="NOP" />
      </MExpr> </AExpr> </AExprR>
    - <MExpr>
       <Const unOp="NOP" value="1" />
      </MExpr>
     </AExpr> </RExpr> </LExpr> </Expr>
    </ParExpr> </MExpr> </MExprR>
   <VarRef _id="id20" var="id8" unOp="NOP" />
  </MExpr> </AExpr>
 <VarRef _id="id1b" var="id6" unOp="NOP" />
</DiffEquation>
```

Berkeley
University of California

So what, again?

- Why was XML chosen in the first place?
  - To take advantage of the *benefits* within the XML TS
    - Automatic validation
    - Model-based syntax generation
    - Easy serialization to multiple formats/representations with XSL
- This exemplifies how the *set* of technologies that share this syntax/metamodel/user-base of XML can have a sort of synergetic benefit

Which do you like better?

Berkeley
University of California

The HSIF modeling environment (named $HSIF_{GME}$ because of its configuration in the GME domain) is a DSME specially configured for the hybrid systems domain. Users can create hybrid systems models with $HSIF_{GME}$ and generate XML files of those HSIF models to interchange with other HSIF tools. $HSIF_{GME}$ generates artifacts (via UDM objects created according to another HSIF metamodel) in the XML domain (XML files), which conform to the HSIF.xsd schema document as specified by the HSIF standard. Incidentally, the UDM objects created during the generation phase are guaranteed to produce valid $HSIF_{XML}$ files, because the UDM HSIF metamodel is used to generate the HSIF.xsd schema.

Without TS concepts

The HSIF modeling environment ($HSIF_{GME}$) is a DSME in the GME TS, and is configured for the hybrid systems domain. $HSIF_{GME}$ uses model generators to generate objects in the UDM TS, which provides a convenient interface to serialize those objects in the $HSIF_{XML}$ TS, which uses XML as a storage medium, and is configured by the HSIF.xsd schema. Incidentally, the metamodel of the $HSIF_{UDM}$ TS is actually used to generate the HSIF.xsd schema, which ensures the validity of the final $HSIF_{XML}$ artifact(s).

With TS concepts

Which do you like better?

Berkeley
University of California

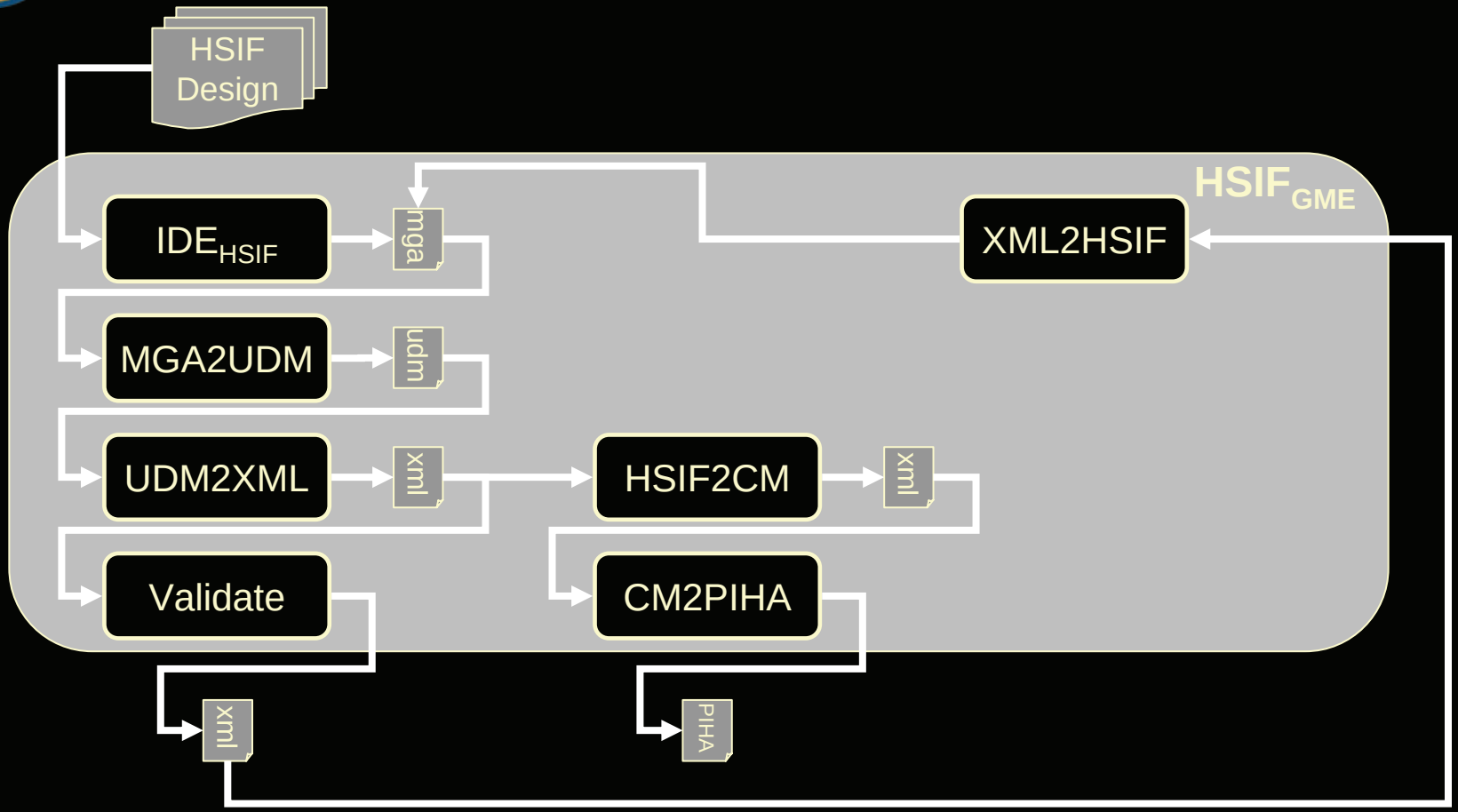
The HSIF modeling environment (named $HSIF_{GME}$ because of its configuration in the GME domain) is a DSME specially configured for the hybrid systems domain. Users can create hybrid systems models with $HSIF_{GME}$ and generate XML files of those HSIF models to interchange with other HSIF tools. $HSIF_{GME}$ generates artifacts (via UDM objects created according to another HSIF metamodel) in the XML domain (XML files), which conform to the HSIF.xsd schema document as specified by the HSIF standard. Incidentally, the UDM objects created during the generation phase are guaranteed to produce valid $HSIF_{XML}$ files, because the UDM HSIF metamodel is used to generate the HSIF.xsd schema.

Without TS concepts

The HSIF modeling environment ($HSIF_{GME}$) is a DSME in the GME TS, and is configured for the hybrid systems domain. $HSIF_{GME}$ uses model generators to generate objects in the UDM TS, which provides a convenient interface to serialize those objects in the $HSIF_{XML}$ TS, which uses XML as a storage medium, and is configured by the HSIF.xsd schema. Incidentally, the metamodel of the $HSIF_{UDM}$ TS is actually used to generate the HSIF.xsd schema, which ensures the validity of the final $HSIF_{XML}$ artifact(s).

With TS concepts

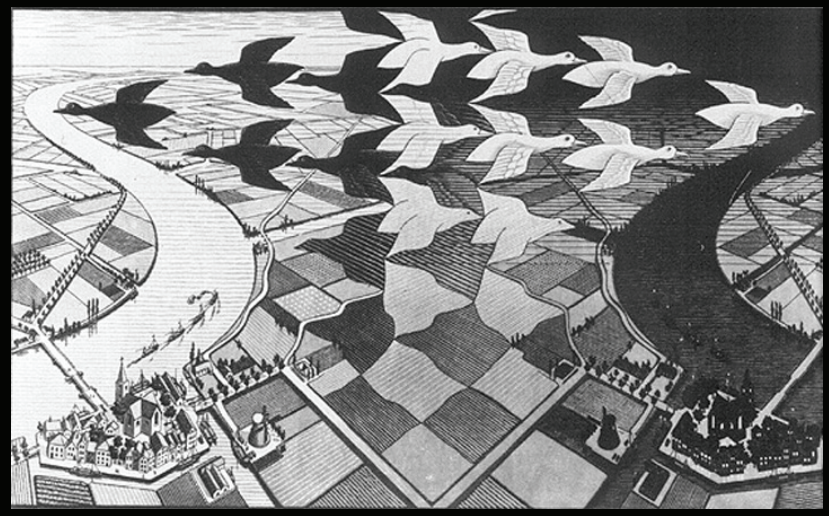
In addition



- *Useful tools produce multiple kinds of outputs!! (and take multiple kinds of inputs as well!!)*
- *This makes my job (as a 'teacher') even harder!*

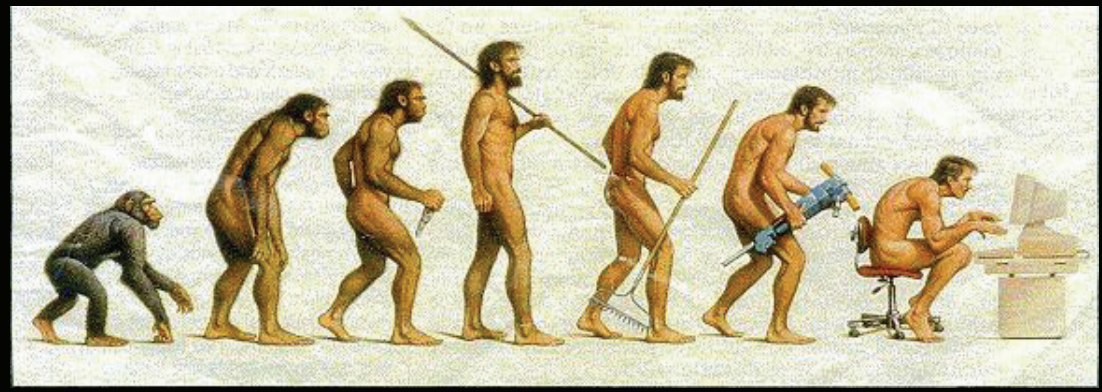
TS Concepts:

- Make the description more precise
- Provide a terse explanation for the *how* (more than the *what*) the semantics are encoded
- Serve as a *guidance* for designers
- Helps DSME designer to more easily explain how identical semantics are encoded using multiple technologies



The Ending?

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.
— T.S. Eliot



- TSs are used mostly as a notational tool for
  - quickly explaining output artifacts
  - studying a design for improved translations within the same TSs
- Quite helpful in removing that "deer in the headlights" look when trying to explain subtle transformations performed by a tool

Berkeley
University of California

Future Ideas

- Thoughts and questions from experts here
- Indoctrination into the DSME discipline as a common descriptive notation
  - Increased usage will clarify weaknesses
  - Expand the symbols to differentiate between 3rd Party tools and transformers for increased understanding of efforts required
- Formalization of concepts for some kind of "generator"??
  - highly unlikely, unless for something like documentation

Questions

"Well HAL, I'm damned if I can find anything wrong with it."
"Yes. It's puzzling. I don't think I've ever seen anything quite like this before."
-- *2001: A Space Odyssey*