Supporting An Architecture-Based Approach to Systems Modeling

John Georgas
Institute for Software Research
University of California, Irvine
Presented at GSAW 2003

Outline

- **Context and Background**
- Techniques and Tools
- Goals and Results
- Open Questions
- Conclusions and Future Direction

Systems Modeling is Difficult

- Why?
  - Usually, there is a large number of different groups involved in the design of any given system.
  - Each group operates in a heterogeneous domain, and may approach their work in different ways.
  - The work of one group may have far-reaching effects on the work of other groups.
- Space exploration mission design is an excellent example.
  - Example groups/domains
    - Control software.
    - Hardware design and construction.
    - Mission cost analysis.

Incongruent modeling techniques, and implicit representations of dependencies and connections are not sufficient.

Addressing Difficulties in Mission Modeling

- CCSDS Architecture Working Group
  - Conceptual goal: Develop overall CCSDS architectural concepts, and establish standard modeling techniques to be used by such organizations as JPL.
  - Practical goal: Provide a designer toolkit supporting and automating these techniques.
- In support of this practical goal…
  - Use ISR/UCI *software architecture* modeling concepts, techniques, and tools to support *mission design* – and, more broadly, *systems modeling*.

Models, even well-defined ones, are of limited use without a firm connection to the realization of what they model.

Outline

- Context and Background
- **Techniques and Tools**
- Goals and Results
- Open Questions
- Conclusions and Future Direction

xADL

- An extensible, modular architecture description language based on XML.
  - Initially, intended for software architecture, so…
    - **Components**, and **connectors** at the forefront.
  - But…
    - Completely modular and extensible using standard XML schemas.
- Why xADL?
  - Established “pedigree” of usefulness.
  - Extensible and modular.
  - Extensive commercial tool support for XML.

Other Tools

- Apigen
  - An automated data binding library generator using the XML schema.
- ArchStudio 3.0
  - A component-based, extensible, software architecture development environment.
    - ArchEdit
      - A generic, context-aware, lightweight editor.
    - Critics
      - Design-time analysis framework and tools.
- Why?
  - A collection of “free” tools and development framework.
  - Context-aware tools that automatically adjust to different models.
    - Quick experimentation with different modeling decisions, essential in this emerging domain.

Though initially intended for software architectures, these tools and techniques are domain independent.

Outline

- Context and Background
- Techniques and Tools
- **Goals and Results**
- Open Questions
- Conclusions and Future Direction

Goals of This Effort

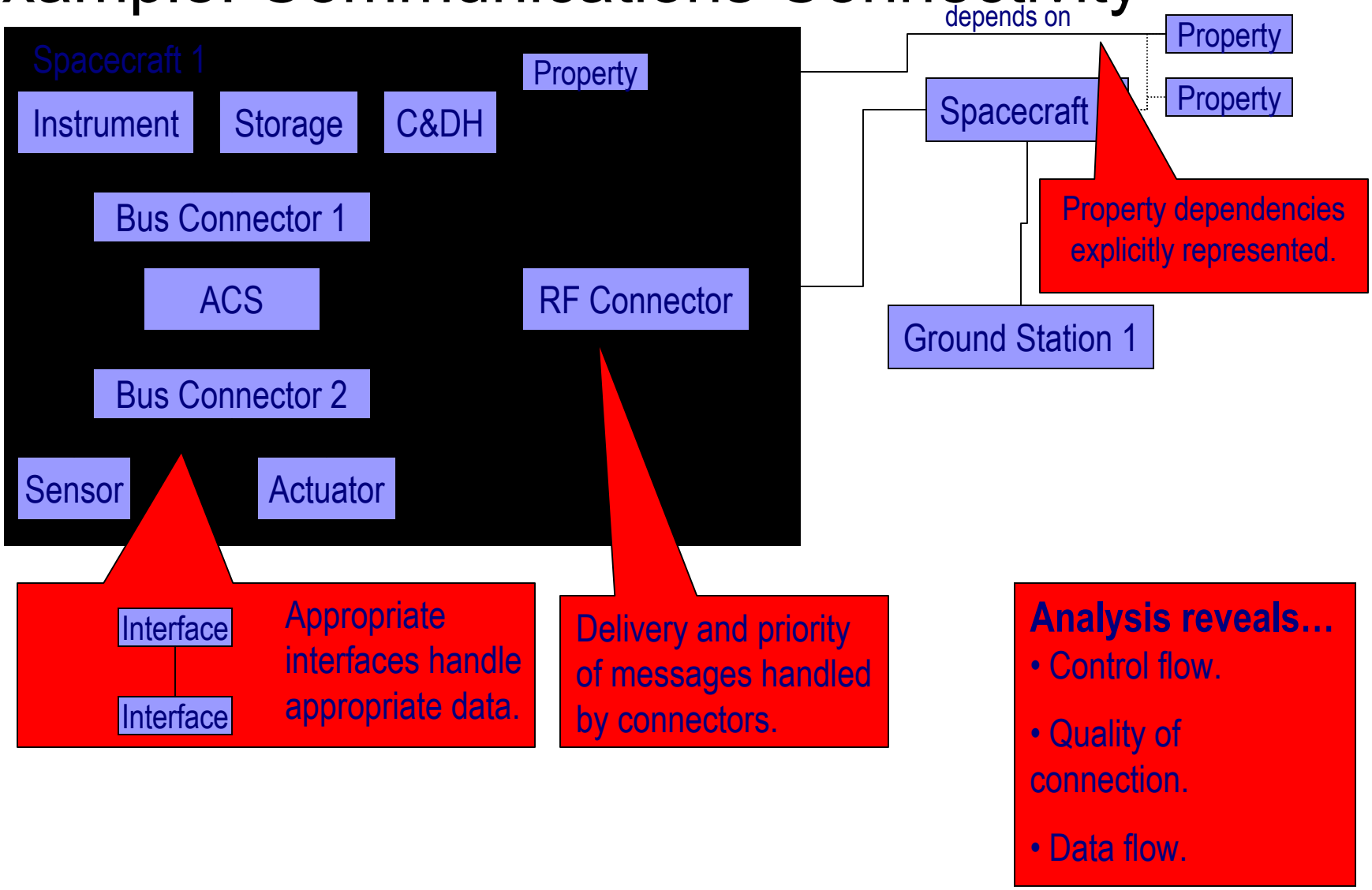
- Determine the usefulness of the ISR/UCI techniques for mission modeling.
  - Task 1
    - Create models (ontologies) of various mission aspects.
  - Task 2
    - Model these as extensions of xADL 2.0.
  - Task 3
    - Explore the usefulness of the base xADL tools.
  - Task 4
    - Provide mission-modeling specific support with tool enhancements.
  - Task 5
    - Provide a simulation framework.

Task 1: Model Creation

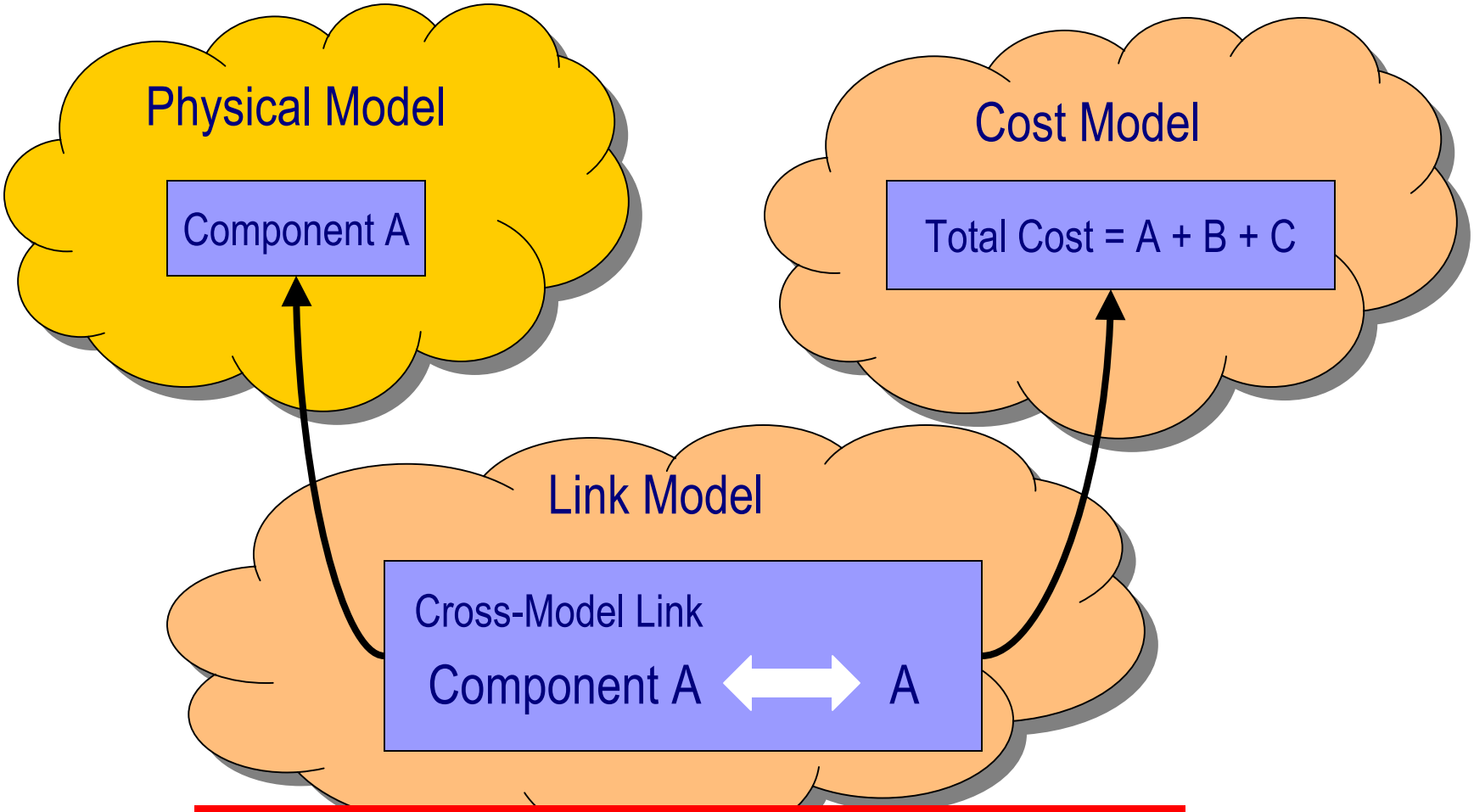
- These models were primarily intended for experimental purposes.
  - Not intended to be definitive.
- Areas modeled included cost analysis, communications connectivity, and control software.

Bringing out **connections** between elements from different **domains** of mission design was a major focus.

Example: Communications Connectivity



Cross-Model Relationships



Task 2: xADL Representation

- Models defined through the use of XML schemas.

```
<xsd:complexType name="MissionStructure">
    <xsd:sequence>
        <xsd:element name="component" type="MissionComponent"/>
        <xsd:element name="connector" type="MissionConnector"/>
        ...
    </xsd:sequence>
    <xsd:attribute name="id" type="archinstance:Identifier"/>
</xsd:complexType>
```

- By using data-binding libraries, architectures can be maintained and manipulated through simple function calls.

```
xarch.add(xArchObjRef, "Component", component);

String id = (String)xarch.get(source, "Id");
```

Other Concepts Represented

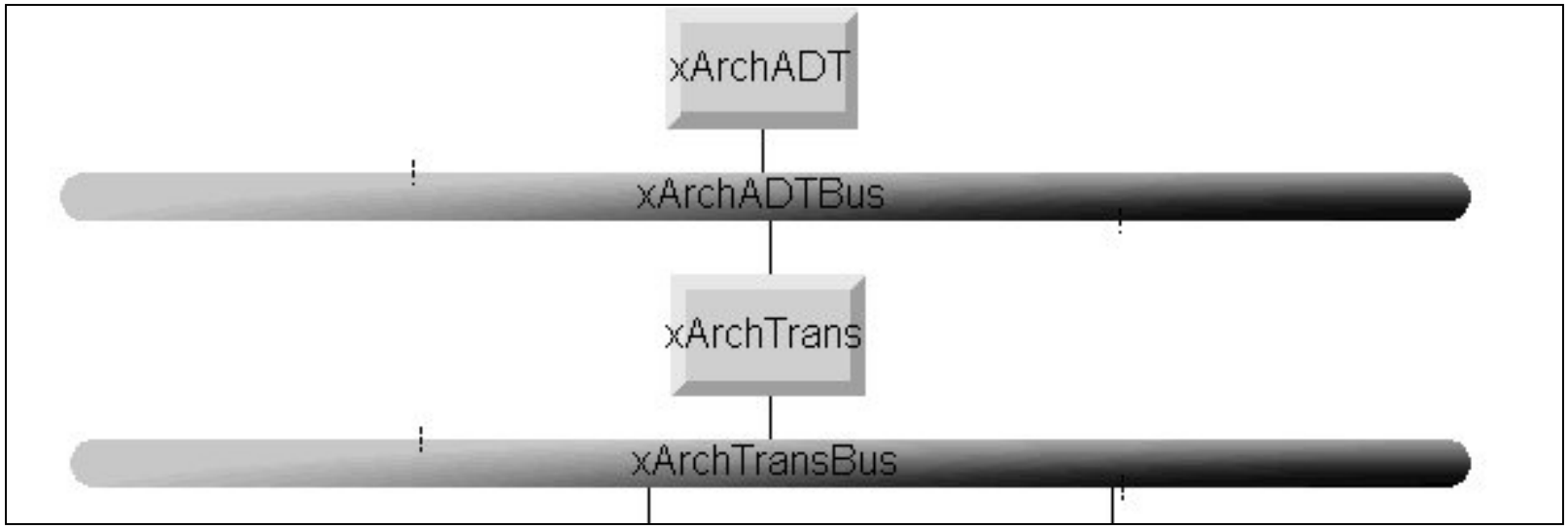
- The ability to specify sub-architectures, and compose models hierarchically.
- Associate properties with components as well as being able to specify derived properties.
- Versioning of all model entities, and the maintenance of version trees.
- Included both instance and type information for promoting reuse.
- The ability to specify mission *temporal evolution* based on environmental properties.

Task 3: Using the Generic Tools

- ArchEdit
  - The editor is context-aware, meaning it can be used with any provided schema with no need for modification.
  - A “free,” high-fidelity editor with every model.

Graphical Depictions

- Microsoft Visio
  - Integrated into the ArchStudio 3.0 tool suite.
  - Event-based coordination between all ArchStudio components, so that the view you see is faithful to the state of the model.
  - A “cheap,” high-fidelity graphical editor.
    - The association between modeling elements and their graphical depictions must be made.



Task 4: Tool Enhancements

- ArchStudio 3.0 Critic Framework
  - Critics, Critic Manager, Critic GUI.
  - Design-time analysis with problem identification and suggested fixes.
  - Extensibility through the addition of custom critic components.
    - Development framework facilitates this.
- Designed and implemented custom *mission modeling* Critics.
  - Design sanity-checking; compliance with model semantics not explicitly represented.
  - Cross-model link maintenance.
  - Presentation of cross-model links, and notifications driven by identified dependencies.
- Tool customization is inevitable!
  - Semantics are specific to a given domain.
- But, this is a relatively “inexpensive” analysis tool framework.

Task 5: Simulations with Ptolemy

- A modeling and simulation framework from UC Berkeley, intended for embedded systems.
  - Java libraries covering a variety of domains.
  - XML based file storage (actually, MoML).
- XML is the key for an initial integration attempt.
  - Custom ArchStudio 3.0 “translator” component that creates the basic framework of a Ptolemy simulation.
  - Significant editing must take place with the Ptolemy tools, as semantics of interaction must be represented.

Outline

- Context and Background
- Techniques and Tools
- Goals and Results
- **Open Questions**
- Conclusions and Future Direction

Open Questions and Challenges

- What is the utility of models without formalisms?
- What should a model that incorporates many varied aspects of a mission look like?
- What should the design environment be like?
- How do we efficiently simulate missions once a formal model of them has been defined?

Conclusions

- Lots of open questions!
- We've shown that xADL, though a *software engineering* technology, is general enough to apply well to the mission modeling domain and *systems modeling* in general.
- A formal **syntactic base** can be provided to any set of architectural concepts making any model more than just a collection of graphs.
- Significant **semantic capabilities** can be enabled such as automated design-time sanity checking and dependency management.

Extra Slides

A Quick Primer on Architecture

- The structure of a system's components and how they're connected.
- Behavior as collaborations between these components.
- Constraints on particular details of compositions usually loosely enforced by architectural styles.

Difficulties Abstracted

- Complexity.
- Visibility.
- Changeability.
- Inter-component dependencies.

What's the best way to address these difficulties?

Utility: Boxes and Arrows Only?

- An example: “Pure” UML.
  - Essentially nothing more than imposed conventions on a set of graphical widgets.
- Is it useful?
  - Helps the designer achieve clarity of vision.
  - Allows for communication of ideas from a common context.
- **But**,
  - How close is the implemented system to the model?
- How do we achieve a tight coupling of implementation and model?
  - In the software engineering world, one can generate source code from formally specified UML models.

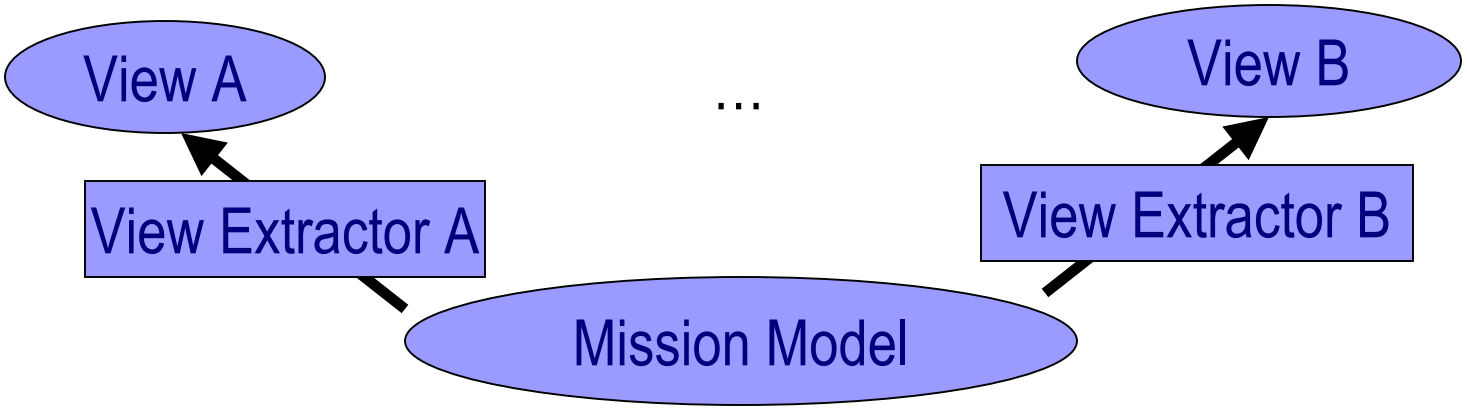
What does this “generative” approach mean in the mission modeling world? Is the utility of a formal approach limited to design-time?

Open Questions!

- What is the utility of models without formalisms?
- **What should a model that incorporates many varied aspects of a mission look like?**
- What should the design environment look like?
- How do we simulate missions once a formal model of them has been defined?

Model: Unified

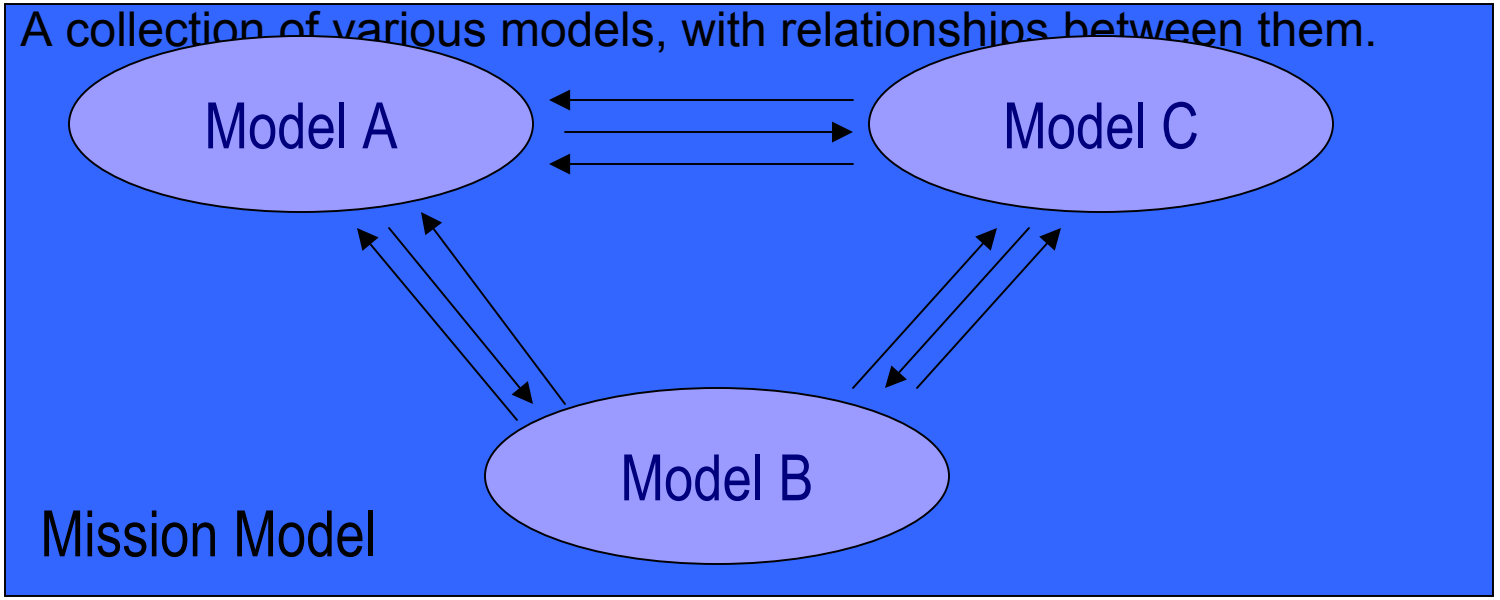
- Unified Model
  - Central representation with various views.



- Why?
  - Coordination
  - Version management
- Why not?
  - View extractor trust
  - Visibility of changes

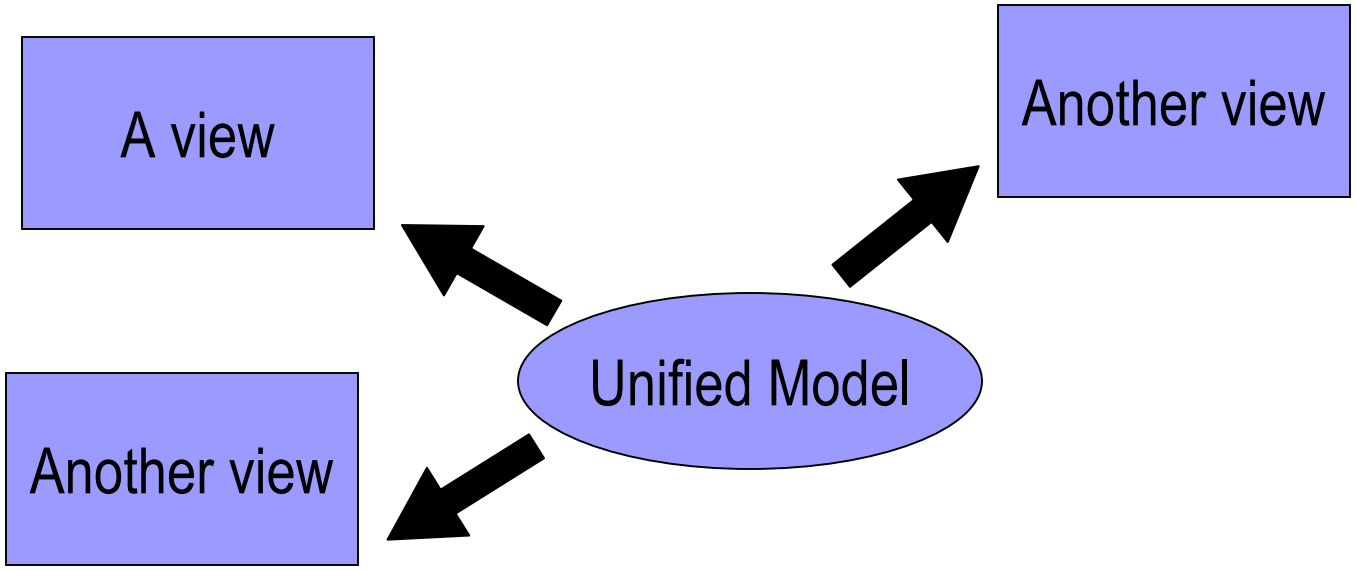
Model: Not!

- Decentralized Model
  - A collection of various models, with relationships between them.



- Why?
  - Version management.
  - Clarity of each model.
- Why not?
  - Cross model knowledge.
  - Large number of links.

Model: Are They The Same?



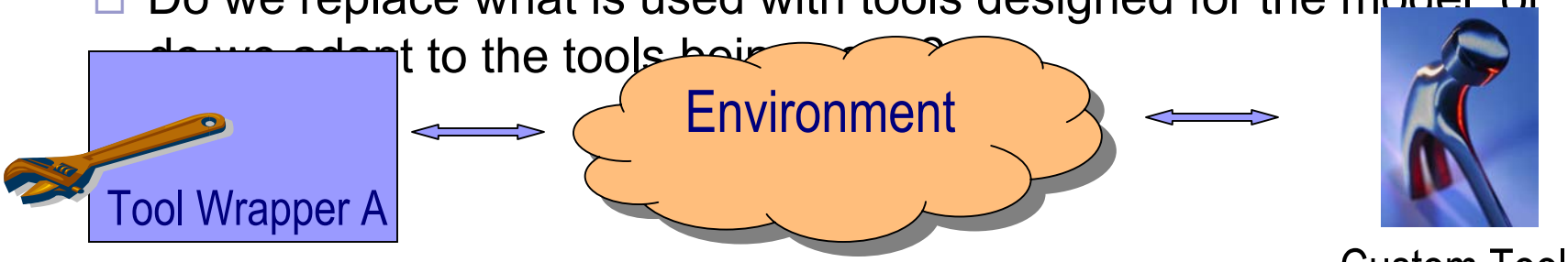
Questions: Which one is better for mission design? Is a central model even possible?

Open Questions!

- What is the utility of models without formalisms?
- What should a model that incorporates many varied aspects of a mission look like?
- **What should the design environment look like?**
- How do we simulate missions once a formal model of them has been defined?

Design Environment: Is Change Good?

- The Big Question
  - Do we replace what is used with tools designed for the model, or do we adapt to the tools being ...?



- Why?
  - Little changes for the users other than enhancements.
- Why Not?
  - Some semantics and model-enabled capabilities may be lost.

- Why?
  - Fully functional with preservation of semantics.
- Why Not?
  - Costly to build.
  - Resistance to adoption.

Questions: Do we change the way people do work? What if it buys us a lot?

Open Questions!

- What is the utility of models without formalisms?
- What should a model that incorporates many varied aspects of a mission look like?
- What should the design environment look like?
- **How do we simulate missions once a formal model of them has been defined?**

Simulation: Let's Build It!

- What do we need to do…
  - Define and code a model of a mission environment.
  - Define and code the behavior of an element within this environment.
  - Define and code interactions between elements, and between elements and their environment.
  - Capture results, and organize them in an understandable way.
- Why?
  - Domain knowledge.
  - Expressiveness.
  - We know how to code.
- Why not?
  - A lot of development.

Simulation: Let's Reuse It!

- What do we need to do…
  - Define a mapping from our model to the simulation framework.
- But,
  - Is the simulation framework domain independent?
    - If not, how do we deal with the semantics of the simulation framework?
  - Can we automate the mapping process?
    - If not, we'll have to "code" anyway, but this time in what may be an unfamiliar environment.
- Why?
  - Reuse of the framework.
- Why Not?
  - See questions.

Question: Which one is the best balance between cost and expressiveness?

Outline

- Motivational Forces
- Quick Introductory Topics
- Results
- Open Questions
- **Conclusions and Future Direction**

Where To Now With This Approach?

- A clear vision of a desired model must be established and accepted.
- A formal definition of this model can then be created using xADL.
- With the formal definition in place, a basis for a mission design environment is in place due to the generality of the xADL tools and ArchStudio 3.0.
  - An easy way to maintain model-definition documents.
  - A “free” low level editor.
  - A framework for a modular design environment is already in place.
- Custom tool design
  - A front-end graphical editor that has a bit of domain knowledge.
  - Model maintenance and analysis tools.
  - Tools that connect the mission model with the actual implementation.