A COTS INFRASTRUCTURE FOR DOMAIN SPECIFIC DIAGRAM SEMANTICS

R Balzer

Rapporteur: Avelino Zorzo

A COTS Infrastructure for Domain Specific Diagram Semantics

Bob Balzer

ISI

balzer@isi.edu

Abstract (1 of 2)

We have extended PowerPoint to provide a COTS framework for enabling external programs to incrementally track and respond to changes being made by a user to a diagram. The responses can:

- annotate the diagram with domain specific errors, suggestions, and/or analysis results
- make further changes to the diagram which compensate for, or are derived from, the user's changes (ala Sketchpad)
- make changes in the external environment to reflect the changes and state of the diagram (i.e. the diagram is a controller for the external environment)

PowerPoint provides the graphic user interface, editing engine, and persistent storage for the diagrams being constructed. Instrumented Connectors provide the means for monitoring user changes to these diagrams.

Abstract (2 of 2)

A protocol communicates these changes to external programs so that they can track and respond to them. This protocol also allows those programs to annotate the diagram and/or make further changes. Both parties can create atomic transactions so that "intermediate" states are neither analyzed nor displayed. It also provides synchronization mechanisms so that these external programs can track the user's focus as it changes among diagrams in the same or different documents.

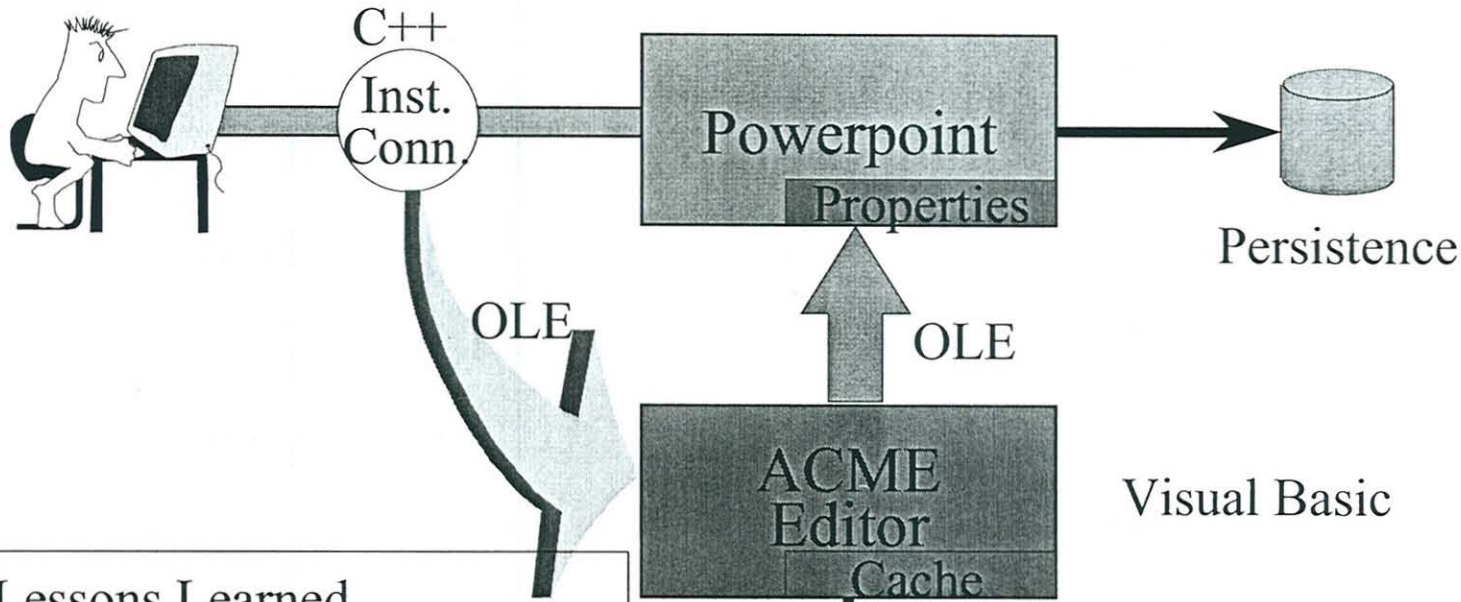
Three different graphic design and analysis domains have been created using this infrastructure:

- Software Architectures
- Satellite Networks
- Survey Authoring

They will be demonstrated in this presentation.

Architecture Editor Demo

ACME Architecture Editor



Lessons Learned

- PC COTS Functionally Rich
- PC COTS Highly Adaptive
- Inst. Connectors Crucial (Visibility)
- No Architecture Mismatch

Effort to Build

- Elapsed Time — 2.5 Weeks
- Development Time

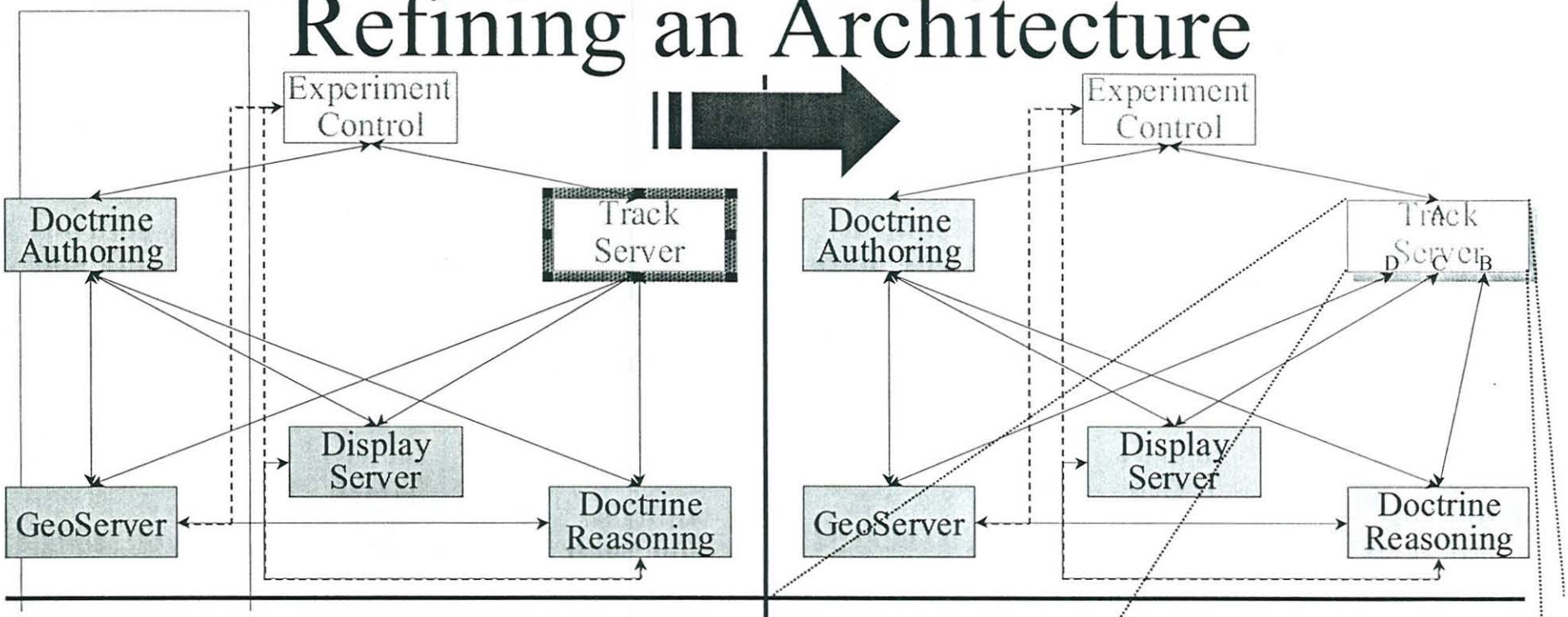
– PowerPoint Driver	7	Days
– Instrumented Connectors	1	Days
– ACME Analyzer	5	Days
Total	13	Days

- Code Size

– PowerPoint Driver	15	Pages
– Instrumented Connectors	2	Pages
– ACME Analyzer	10	Pages
Total	27	Pages

Multi-Level Architecture Demo

Refining an Architecture



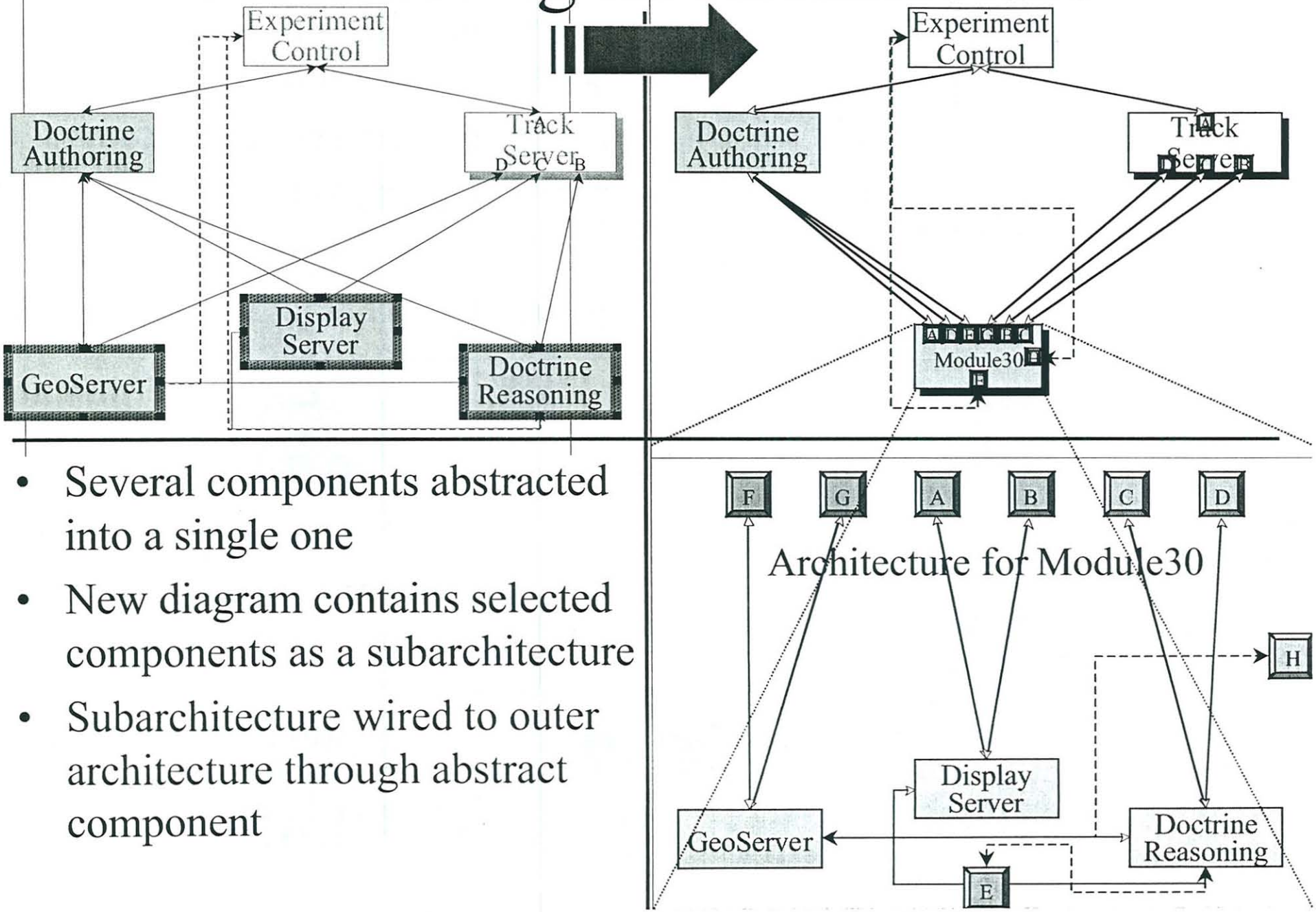
- New diagram contains (empty) subarchitecture for selected component
- Subarchitecture wired to outer architecture through refined component
- Shadow on refined component indicates it has a subarchitecture

A

Architecture for Track Server

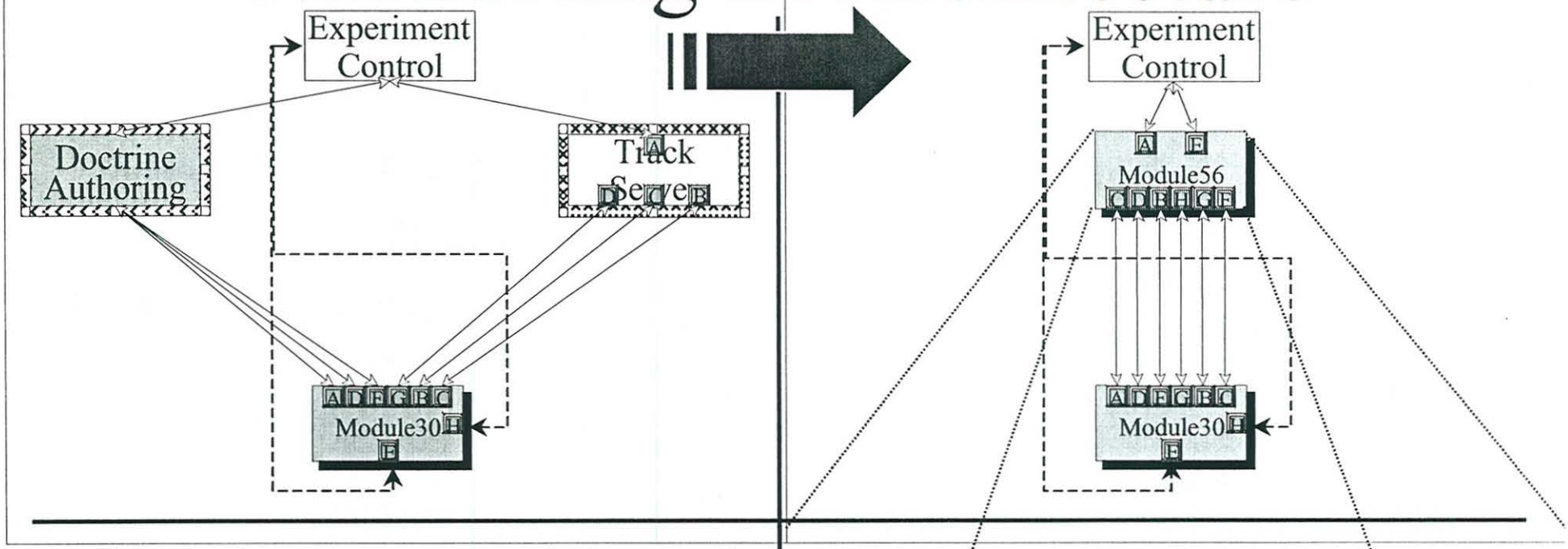
B C D

Abstracting an Architecture

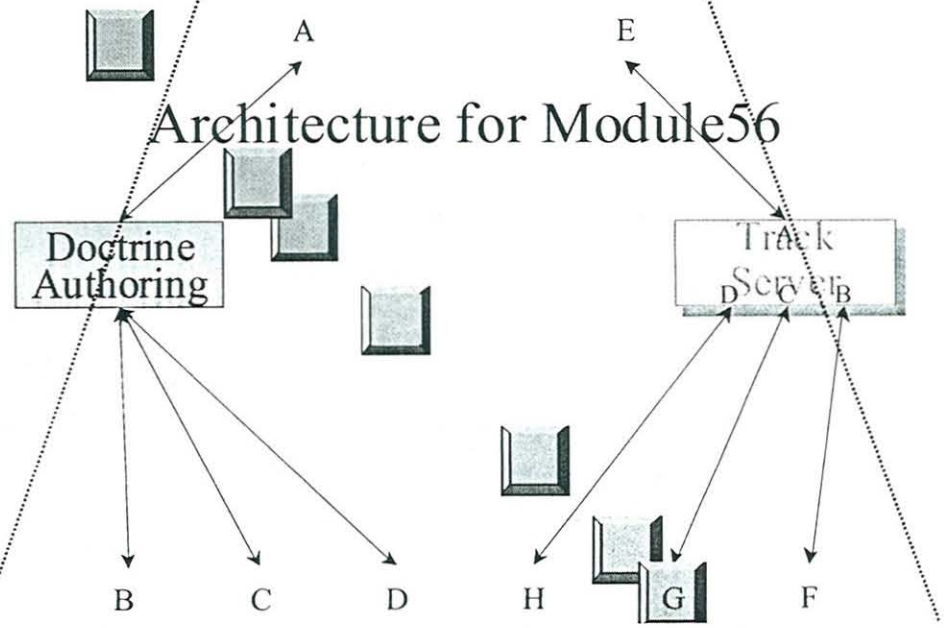


- Several components abstracted into a single one
- New diagram contains selected components as a subarchitecture
- Subarchitecture wired to outer architecture through abstract component

Abstracting an Architecture

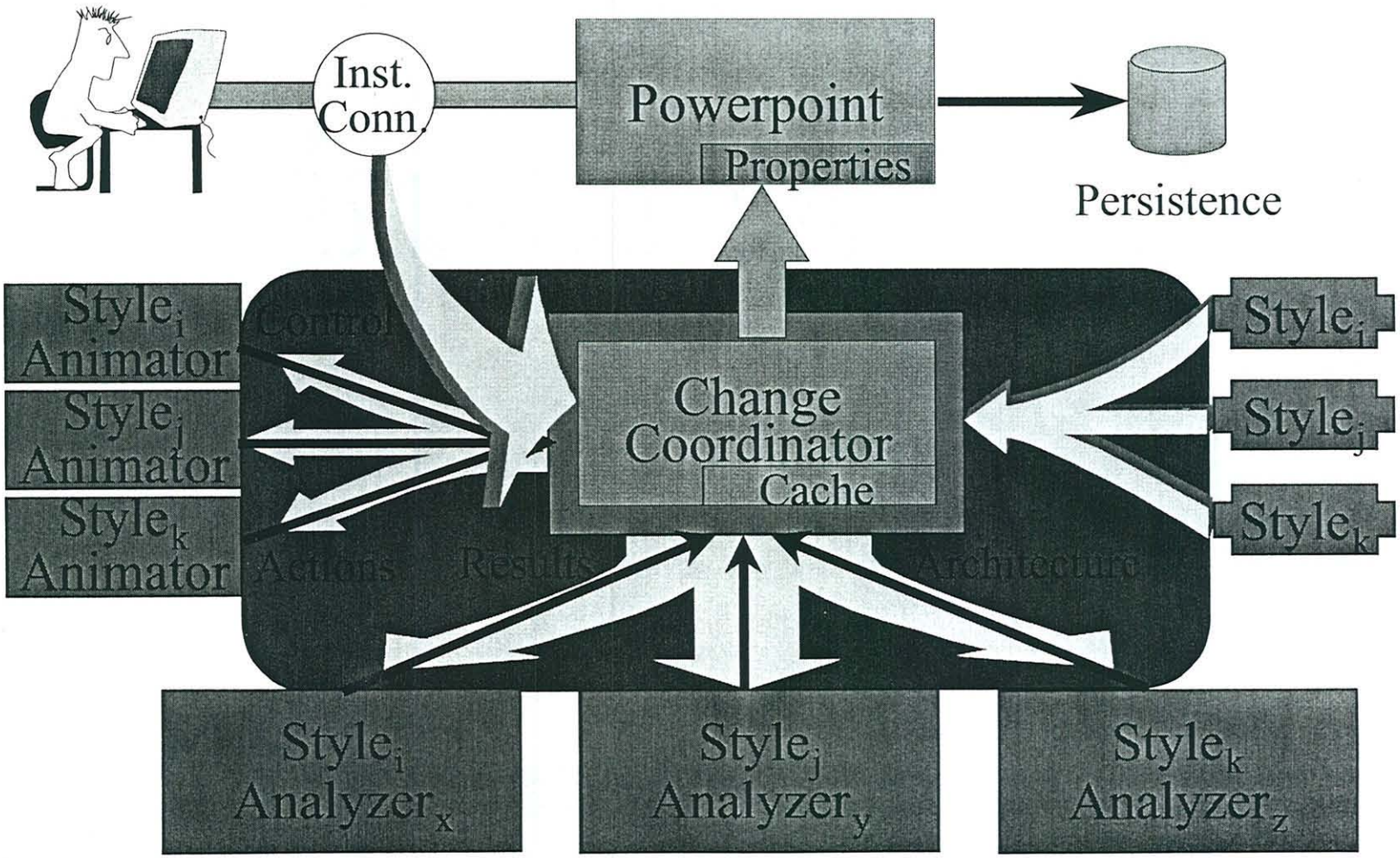


- Several components abstracted into a single one
- New diagram contains selected components as a subarchitecture
- Subarchitecture wired to outer architecture through abstract component



Domain Specific Diagram Semantics Demo

Domain Specific Diagram Editor



Domain Specific Diagram Animation Demo

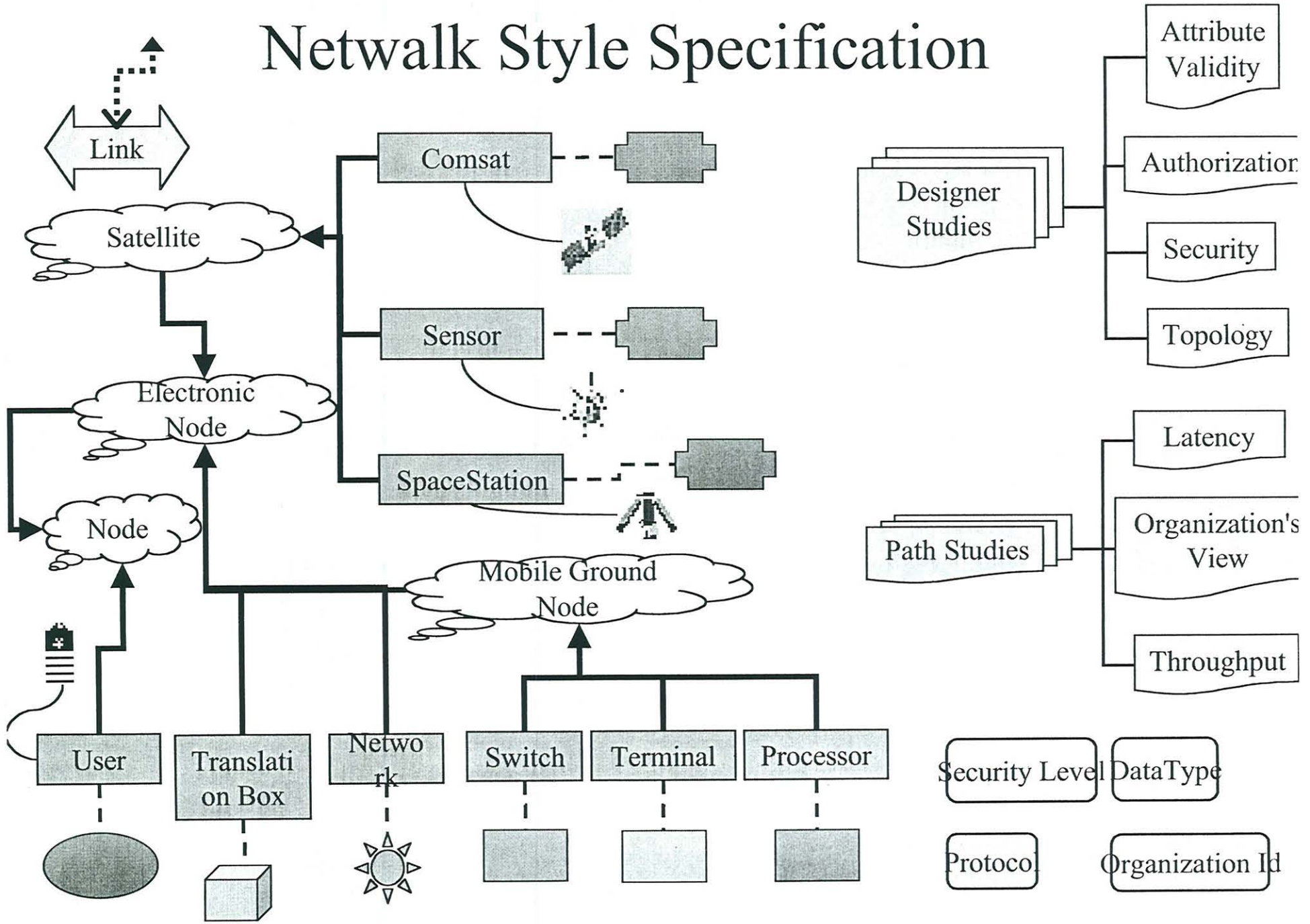
Features

- COTS Graphic Editor
  - Components & Connectors
  - Persistence
  - Multi-Level Architectures
    - Graphical Refinement
    - Graphical Abstraction
- User Defined Styles
- Interactive Style-Based Analyses
  - Incremental or Snapshot
- Interactive Style-Based Behavior Animation
  - Simulation or instrumented execution

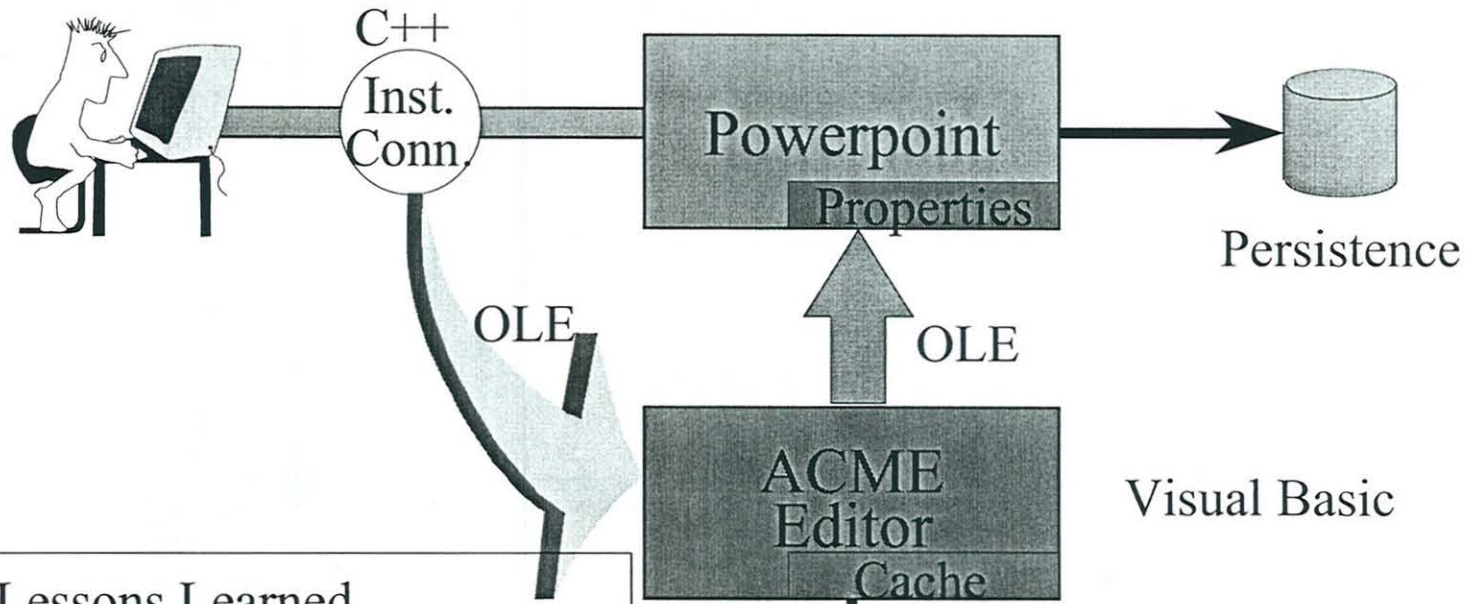
Value Added

User Defined Diagram Semantics Demo

Netwalk Style Specification



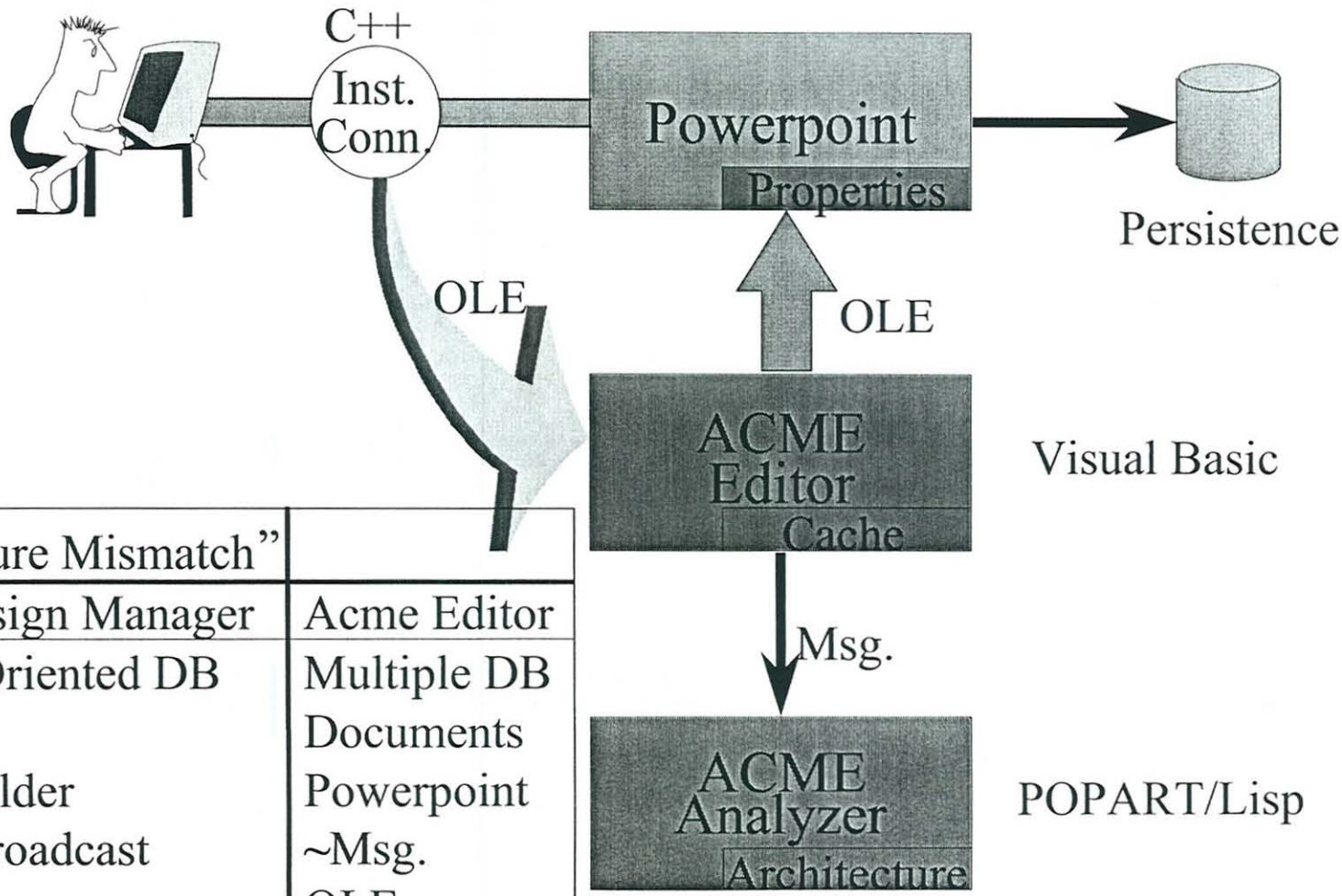
ACME Architecture Editor



Lessons Learned

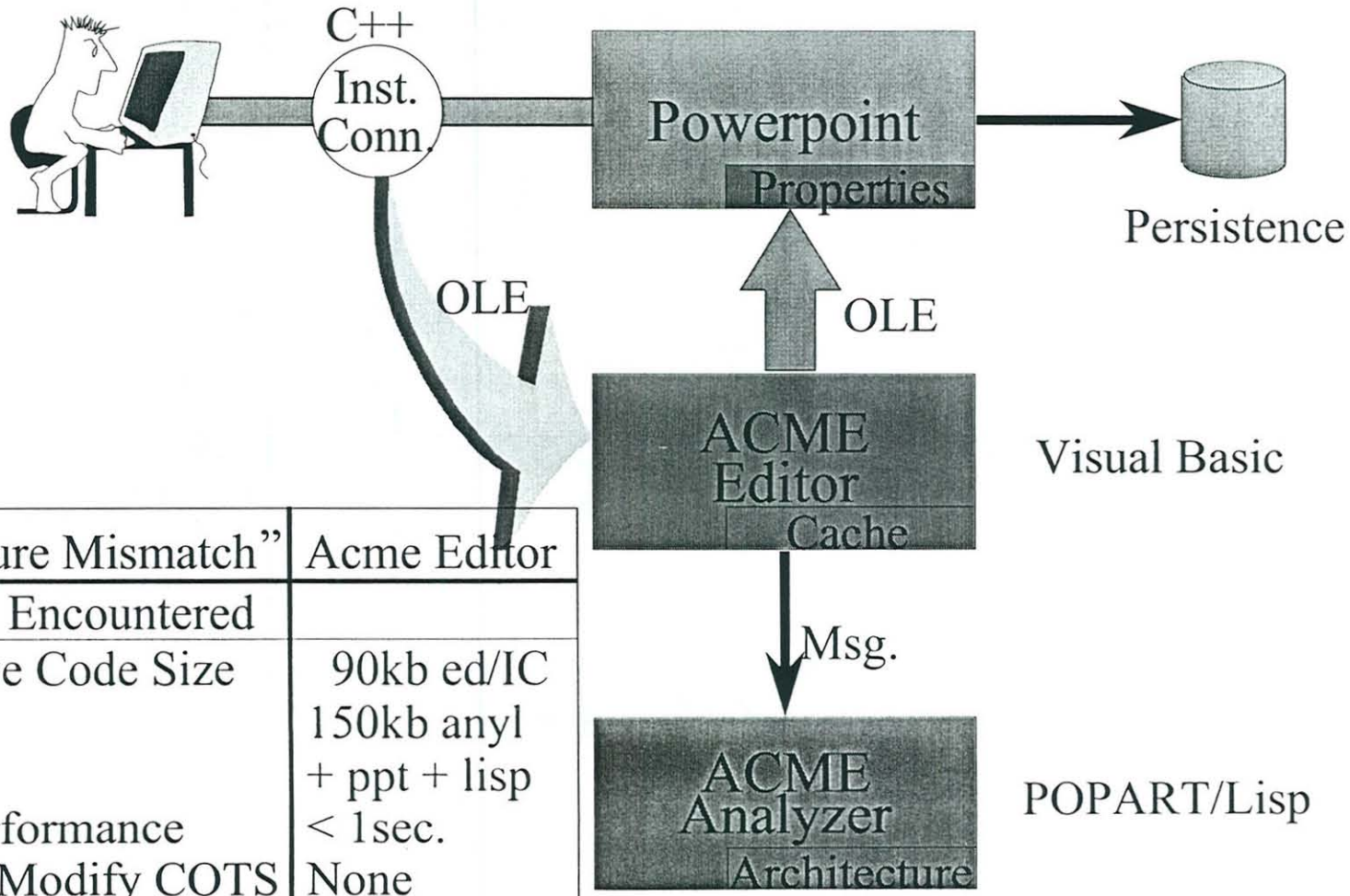
- PC COTS Functionally Rich
- PC COTS Highly Adaptive
- Inst. Connectors Crucial (Visibility)
- No Architecture Mismatch

ACME Editor



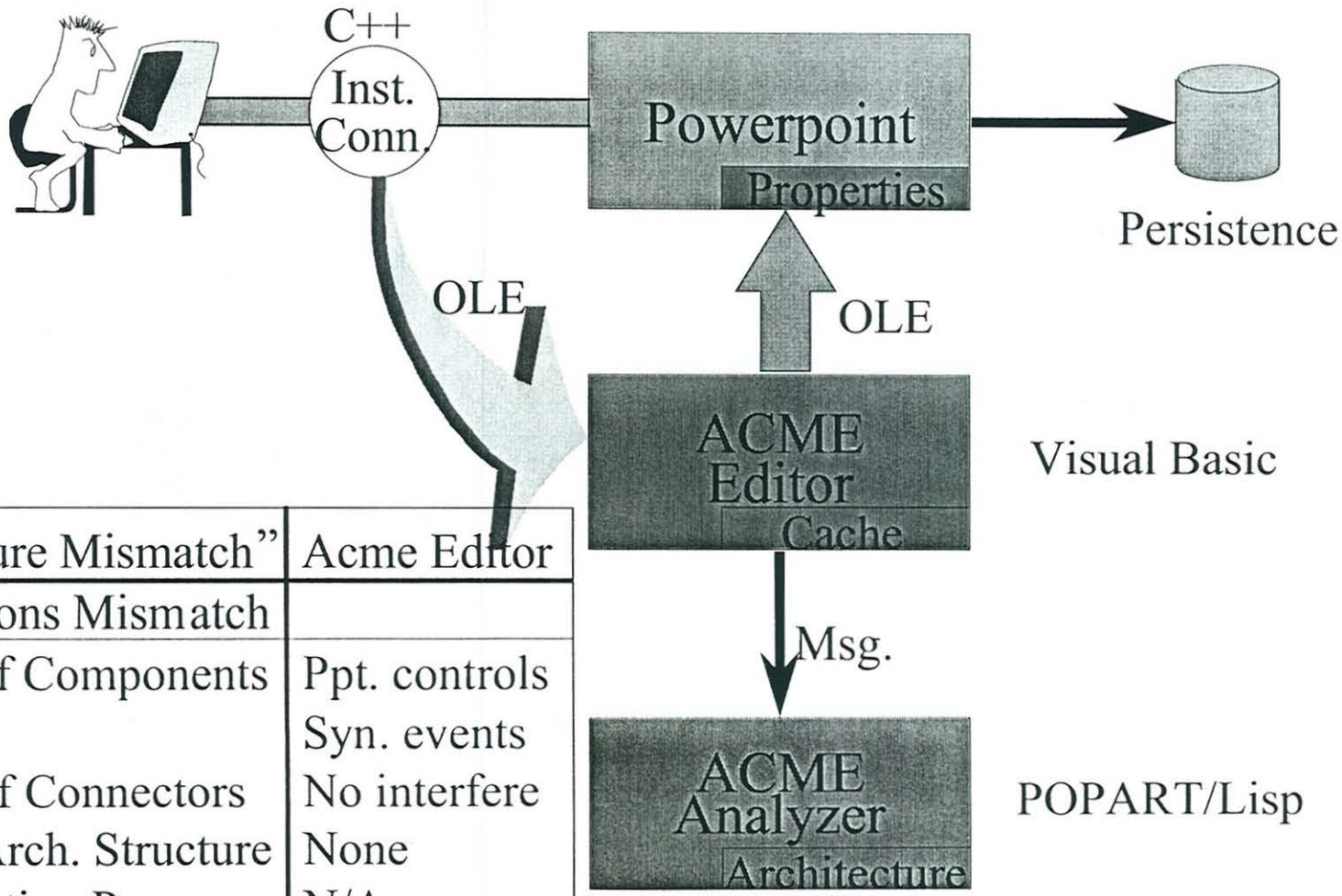
"Architecture Mismatch"	
Arch. Design Manager	Acme Editor
• Object Oriented DB	Multiple DB
	Documents
• GUI Builder	Powerpoint
• Event Broadcast	~Msg.
• RPC	OLE

ACME Editor



"Architecture Mismatch"	Acme Editor
Problems Encountered	
• Excessive Code Size	90kb ed/IC 150kb anyl + ppt + lisp
• Poor Performance	< 1sec.
• Need to Modify COTS	None
• Error Prone "Make"	N/A

ACME Editor



"Architecture Mismatch"	Acme Editor
Assumptions Mismatch	
• Nature of Components	Ppt. controls Syn. events
• Nature of Connectors	No interfere
• Global Arch. Structure	None
• Construction Process	N/A

DISCUSSION

Rapporteur: Avelino Zorzo

Lectures One and Two

During his talks Professor Balzer presented several live demonstrations of his approach for mediating interactions between different software components. In his first talk he mentioned the importance of the approach with some demonstrations, and in the second talk he concentrated on the demonstration of the inclusion of extra capabilities for PowerPoint.

He started the presentation inquiring about how to debug the connections between several components (software). Promptly a member of the audience asked what Professor Balzer meant by debugging an architecture? Professor Balzer answered saying that it is done by checking runtime structures, the ones that implement the design architecture. Professor Balzer also said that it is important to know if an architecture is doing what it is supposed to do, so it is desirable to be able to observe a certain set of things: control flow, the intermediate values that are being computed, and so on. Traditionally this is done by debugging using tracing techniques, inserting breakpoints, ... Another point is the possibility of inserting faults in the components to check if they are really robust. Professor Balzer mentioned two kinds of faults that one might be interested in checking: architecture faults, if the connections between components are functioning the way they are supposed to; and environment faults, if the component is performing according to a set of expectations.

To perform this type of checking is not easy because components are usually black boxes (written in several different languages), Professor Balzer said. One cannot check the inside of components, but one can know what is happening in the architecture by checking what is happening in the connectors of the component. Professor Balzer pointed out that we have to focus on technologies that allows us to see what is happening in the architecture level, on the interactions that happen between a variety of different kind of connectors. Professor Balzer mentioned that he has an abstraction of architecture connectors, which is a conduit for all interactions that occur between components modules, e.g. network sockets, remote procedure calls, CORBA, etc. He is working on a technology that allows the insertion of a program in the middle of that connector. The program is an arbitrary program that can do things like instrument and check the data that is flowing between components, e.g checking data, inserting additional computation.

Professor Shaw asked if one should do this. Professor Balzer answered it later, when discussing one of the examples, saying that one might want some form of protection for his data, so he could leave this data encrypted in the file system, and insert a program that would decrypt the data for someone with the rights of accessing that data.

Professor Balzer inquired of the audience about what one could see about an architecture, and then mentioned that if all we have is an API of a component, then we can see some of its behaviour. On the other hand if you can see the operating system interface that the component has to the system, then you can see more. If you can see the connectors the component has with other components, then you can see even more. If you can see the graphical user interface of the component, then you can see the entire behaviour of the component.

Professor Balzer said that, basically, Windows and Unix, are built around Dynamic Load Libraries (DLLs). So several services (operating system, network services, CORBA, ...) are packages provided by a manufacturer. Components use some modules of those libraries. The approach Professor Balzer presented was a way of pasting an extra program in front of particular elements. The only requirement was that the program (mediator) had to have the same API. Professor Balzer said that this becomes very powerful. In both Unix and Windows, the designers have decided to pack everything has a DLL. Professor Balzer mentioned that he can mediate operating system services, network services, and so on.

Professor Balzer said that this is not a new idea. It is an old idea, and the earliest he could recall about this idea was from an advice system in LISP in the early seventies. That system would allow you to put your own code around functions calls. In LISP everything was a function.They used this for debugging or for creating extra capabilities.

Professor Randell asked if at the heart this was not what nowadays is called reflection. Professor Balzer did not agree with that. Professor Randell then said he was not saying it was reflection, but it could be part of the mechanism that is being used for achieving reflection. Professor Balzer said then if there was reification then this certainly would provide the means for doing that.

From this point on, Professor Balzer presented a set of examples of using his approach.

1. Professor Balzer first used Microsoft Word to show what calls are used by the editor. He showed a mediator that would allow, prohibit, or check around 460 calls to the Windows kernel. During the demonstration Professor Balzer, using the mediator, checked the allocating memory service (global, local and virtual memory allocation). The mediators were placed between the application (Microsoft Word) and the operating system (Windows). When Word was made active on the screen, a bunch of allocation calls was used by Word, and the mediator showed what were the allocations Word asked the operating system. Professor Balzer showed also what would happen when some services of Word were used (it was possible to see, in the mediator, that Word was using the services for allocating memory). Professor Balzer pointed out that one could prohibit the allocation of memory from Word, and the result would look as if the machine was overload.

2. The second application Professor Balzer started to demonstrate was the Microsoft C++ Development Environment. He compiled a small program with 2 errors on it. During the demonstration of this environment Professor Balzer faced some technical problems and was not able to finish his demonstration. Professor Balzer mentioned that the idea was to show the interaction between the components of the Microsoft C++ Development Environment, e.g. the Compiler talking to the Environment, and so on.

3. Professor Balzer said that he also has a spy program that allows him to know which interfaces a program is using, and if wanted he could also extend the architecture interface. The third application of a mediator he presented was an encryption archive system. He used Netscape to access some files from a directory on his computer disk. But what he could see was just a table of contents and an archive. The archive contained all files in an encrypted format. So he could not access the files. Between the Netscape and the File System there is a connector that is a piece of instrumentation that allows you to see the files in the archive, as if they were part of the file system. Professor Balzer said that he could do the same thing for the Emacs, but this would not be interesting because it is not graphical. He showed the same kind of access using Windows Explorer. In reality the files that Explorer was showing were not in the file system, but the Explorer believed they were there. Professor Balzer said that basically he created a virtual file system.

Professor Balzer pointed out that the same idea could be used to integrate COTS products. As an example he mentioned that he has integrated Eudora and Emacs using this approach.

4. The fourth live presentation made by Professor Balzer was to show the integration of Netscape navigator and Microsoft Access. Netscape was used as a tool for collecting information (a survey) and the data was inserted in a data base, using Access. Professor Balzer asked Professor Randell to fill in the survey. After the form had been completed and the data submitted, the mediator would grab that information and insert it in the database. Professor Balzer showed also that one could insert some kind of logic to check if the data that was being filled in was correct.

A member of the audience asked if Professor Balzer had the documentation for the specification of the interfaces between the components. Professor Balzer said he believed everything was documented.

After an inquiry if someone else decided to have a bright idea of inserting a new mediator where there was a mediator, Professor Balzer said it would work.

Professor Balzer said that this technique could be used for extending COTS products as well as to restrict COTS products. Professor Balzer is interested in safe execution environments. One of the things he is looking at is building an implementation of connectors that are non bypassable. He mentioned that he is very close to having a security manager for Windows NT that is essentially programmable by a user. Professor Balzer explained that basically this safety execution environment put a program inside a cocoon, and looks at all its interactions with the outside. The idea is to make tools like web browsers safer, e.g., avoid information from being taken from your machine. The main interfaces Professor Balzer is worried about are the operating system interfaces, and he mentioned that there are only a few of them. For instance there are few functions that allow you to access files (under a dozen).

When Professor Balzer was saying that operating system interfaces are well documented, but you still have to make sure you have thought all the ways people could find security flaws on them, Professor Randell pointed out that the whole history of people finding security flwws, or at least a very large amount of it, is by people realizing the implications of things that were thought not possible to do.

Professor Balzer said that the advantage of using his approach is that the control is outside the operating system, and someone does not need to wait for the manufacturer to go and fix it. For example, if you find there is a problem somewhere you can disallow the use of that service until you have found a way of solving it. The point Professor Balzer was trying to make is that it is under your control, so you can do whatever you want. At this point Mr Jackson mention that a very conservative way of doing this would be not to allow a program to use anything, and then start to allow things progressively.

5. The last demonstration Professor Balzer presented was to show how he extended Microsoft PowerPoint. His PowerPoint has extra functionality, and he showed some examples of this extra functionality, e.g. topology analysis. Professor Balzer mentioned that compared to PowerPoint what was included is tiny. Basically PowerPoint provides the syntactical part of a domain and the analyser connected to the PowerPoint includes some semantics to it, e.g. satellite domain, survey domain, data flow animation.

Professor Balzer said that some of the COTS interfaces were very well documented, and everything a user can do, a program can do as well.