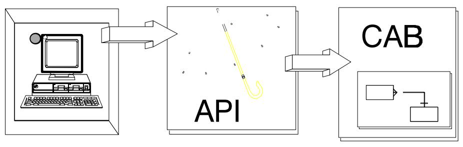
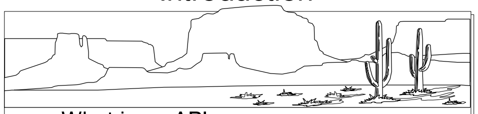
Designing Open API Modules from Composer™ Action Blocks



Presented by:
Ron Johnston
Johnston McLamb CASE Solutions, Inc.
4443 Brookfield Corporate Drive, Suite 105
Chantilly, Virginia 22021
(703) 502-0901

JMCS

Introduction



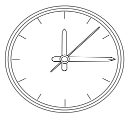
- What is an API
- More powerful component-based development
- Open client/server strategies
- API Wrapper module design
- Keys to mixing OO with Composer

What is an API

- Application programming interface
- Callable submodule
- Could be Client DLL or Server dynamic call
- Black box encapsulation approach
- Great for system bridging or re-use

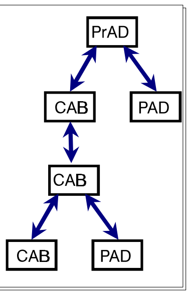
API Concepts

- APIs used in operating systems
- Promotes rapid re-use of tested functionality
- Parameters are "message contracts"
- Internal logic & data is hidden
- Easy, rapid maintenance possible



APIs using Composer

- Goal is to broaden use of Composer logic
- Open up calls from non-Composer requestors
- Tightly integrate Composer & non-Composer applications
- Strong alternative to DDE transaction link

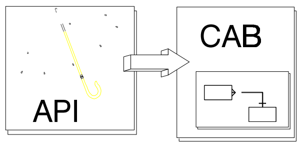


Composer Code Issues

- Composer-generated code issues
  - Global data area (GDA)
  - IEF command field
  - Exit state messages & properties
  - Import & Export Views
- Need to buffer non-Composer requestors

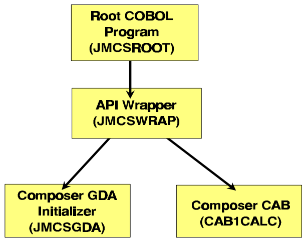
API Wrappers

- Technique used for COTS products
- Add "umbrella" layer to submodule structure
- Composer-specific issues can be hidden
- Wrapper module logic not easy but derivable



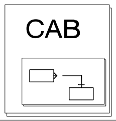
Open API Structure

Example batch COBOL use of API:


Root COBOL Program (JMCSROOT)
API Wrapper (JMCSWRAP)
Composer GDA Initializer (JMCSGDA)
Composer CAB (CAB1CALC)

Open API from CAB

- Any level of common action block can work
- API wrapper based on CAB imports & exports
- Standard GDA initialization needed
- Warning! Proceed at own risk.
- New releases of Composer could cause rework

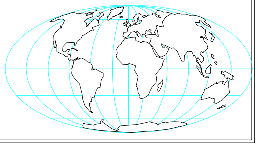


Open API Change Control

- Make message contracts upwardly compatible
- Consider passing expected API version number
- Design in flexibility up front
- Client DLL changes easier if on LAN Servers
- Keep formal inventory of message contracts
- Track API call usage

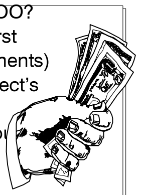
Component Based Development

- JMCS approach is different
- Uses fully encapsulated OO-like components
- Openness of components is key
- API ability an important feature
- More powerful than model-based development



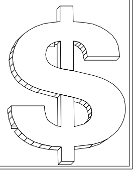
Business Benefits of OO

- Why are Organizations excited about OO?
  - Component-based development (first re-use, then buy, then build components)
  - Behavior of Object part of each Object's definition
  - Minimize Object definition work thro levels of inheritance (ensures consistency)
  - Stability with Flexibility over time (customize only where needed)

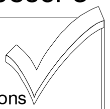


Business Benefits of OO

- Encapsulation minimizes Development Coordination issues
- Rapid maintenance and enhancement of application systems based on internally-built or purchased components
- Containment of Testing Issues
- "Middle-out" approach to Enterprise-Wide Re-engineering



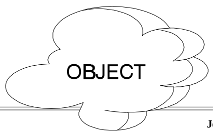
Current Situation with Composer 3



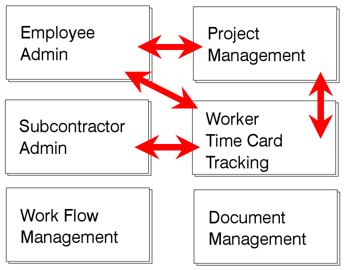
- Great I-CASE tool (perhaps world's best)
- Supports wide range of project sizes
- Able to generate large-scale, integrated, high-performance mission critical applications
- Able to generate for wide variety of computer technologies (hardware & software, GUI client/server, distributed, etc.)
- Can dramatically improve development and maintenance productivity
- Based on IE fundamentals (ERD, PHD, DLG)
- How do we use it today for OO-like results?

Open Service Objects

- Technical or Business subsystems are objects
- Separate Composer models, not migration
- Formal inventory of message contracts
- Growing toolbox of turn-key add-ons

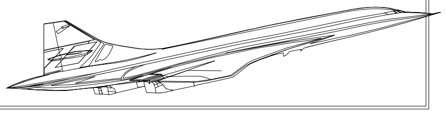


Example Encapsulation



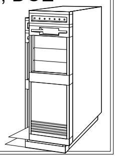
Open Architecture Considerations

- Callable API submodules with examples
- OLE/DDE or DCE links to transactions
- SQL accessible data in RDBMS
- Multi-platform support and connectivity

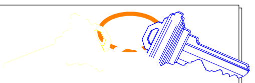


Open Client/Server

- Many possibilities in open architecture
- Client side can use APIs, OLE/DDE, DCE
  - common edit checks, local data
  - office automation integration
- Server side can use APIs, OLE/DDE, DCE
  - real-time bridges or queues
  - cross-subsystem integration
- Asynchronous parallelism also possible



Keys to OO with Composer



- Define large-grained subsystem objects
- At end of BAA1 scope both technical and business objects
- Use multiple isolated models in Production
- OO messaging can't replace SQL

End of Presentation