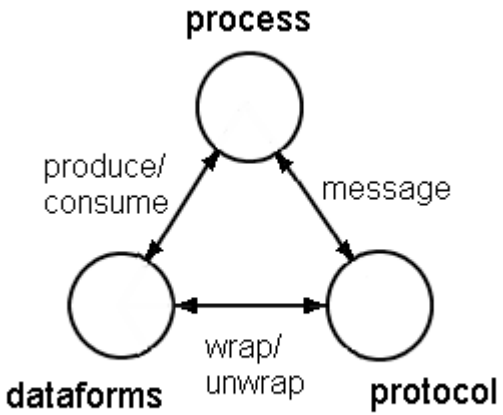
Techniques for Generating Communicating Systems

ICSR7/GP2002

James M. Neighbors
Bayfront Technologies, Inc.
1280 Bison B9-231
Newport Beach, CA 92626 USA
+1 714 436 0322x4
James.Neighbors@BayfrontTechnologies.com

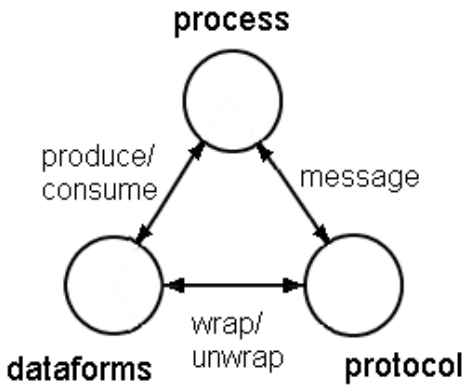
Analysis Model

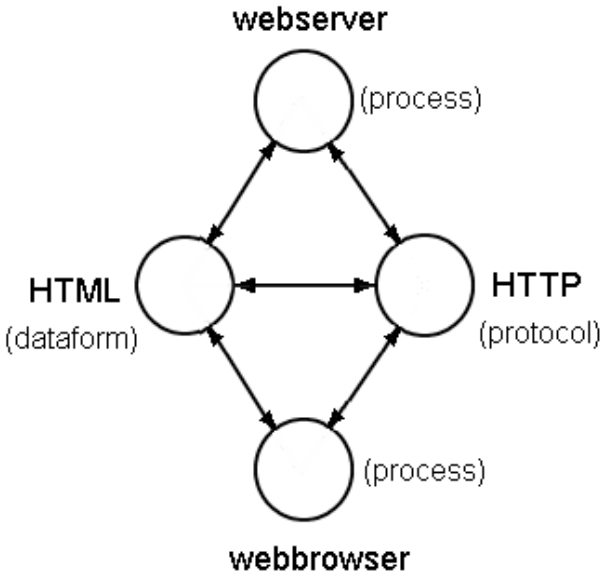


Triad Communicating Systems Model

- Analysis model provides data and control dependencies.
- Processes produce and consume dataforms.
- Dataforms are wrapped, sent and unwrapped by protocols.
- Protocols asynchronously exchange messages with processes.
- This talk is not concerned with details of triad model but with ramifications of using it within a generator.

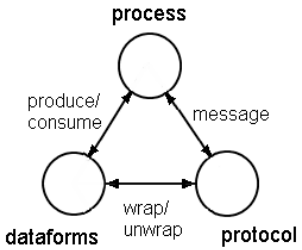
Architectural Model

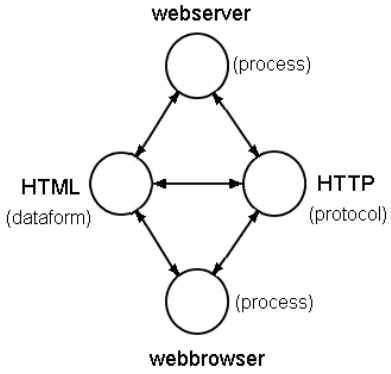


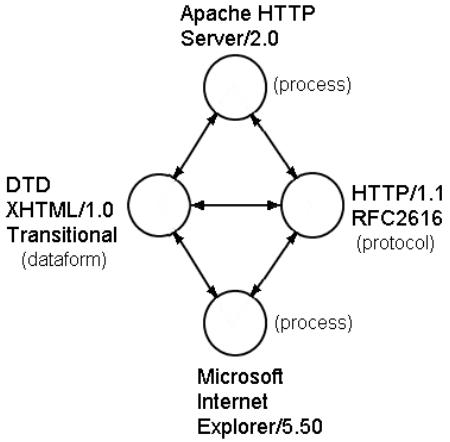


- Architectural model shows abstractions of operating elements.
- Two triads are overlaid to show a web client-server relationship.
- Assume the construction of a generator in this web serving area.
- Level of abstraction (LOA) *decreases* left to right.

Instance Model

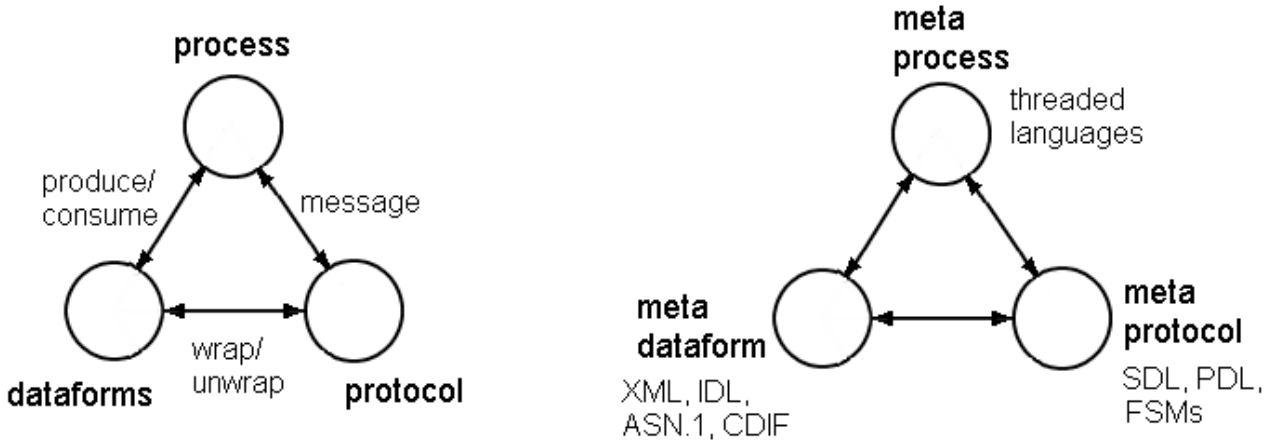






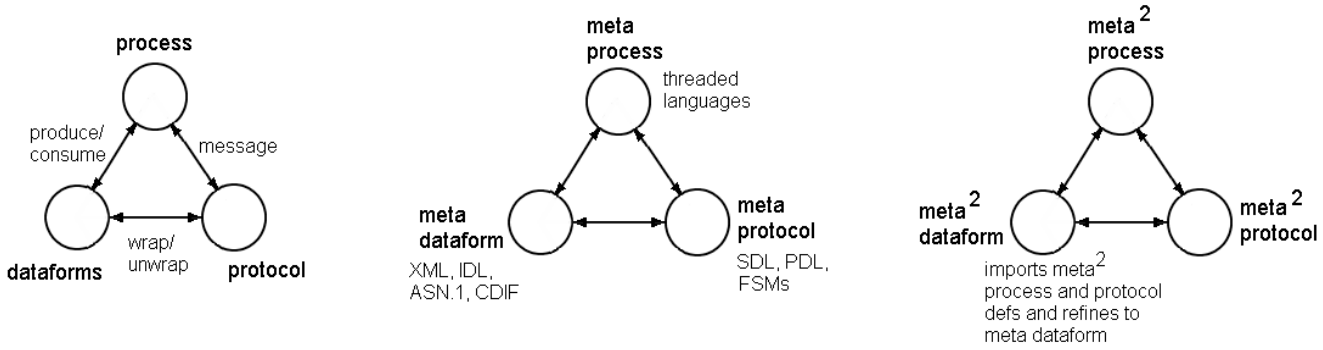
- Instance model shows concrete instances of operating elements.
- Each concrete instance is under the control of a separate social organization.
- Version and configuration space complexity requires Architectural Model be the model maintained by our generator.
- *"The classical defense that generators (and people) have used against low-level complexity is to create a simpler model at a higher level of abstraction."*

Meta Analysis Model



- Level of abstraction (LOA) *increases* left to right.
- As before, meta elements established to cope with lower-level complexity (e.g., what is a complete set of data produced or consumed by a process?)
- We naturally extend basic triad model to meta models of all elements: dataforms, processes, and protocols.
- The society of each analysis element has created their own meta abstractions.

Meta Meta Analysis Model



- Why meta-meta models? As before, meta elements established to cope with lower-level complexity both social and definitional (e.g., versions of XML).
- For generator locates version problems into refinements.
- In some cases definitionally unnecessary (e.g., XML defines IDL, IDL defines XML) but important for composition of domains.
- Deals with “root of the world” objects from databases, networking, OS, applications and graphics.
- These are just implementation domains not application domains.

Conclusions

Isn't having private abstractions anti-standard? *It's good to be standard but not at the expense of trying to maintain an impossible version and configuration space. Where are those "standards" anyway? A generator can be externally standard using internal private abstractions (e.g., subsetting.)*

What is this abstraction process where we are constantly defining a higher level of abstraction to avoid lower-level detail? *In my opinion it is the process of establishing a domain hierarchy under domain analysis. Notice that each of these levels has a domain-specific language, optimizing transformations, and refinements to other domains.*

Why should we not use code fragments directly in the generator? *They have a detail, version and configuration space. Also they can prohibit the production of non-code artifacts such as diagrams, simulators, and formal theory analysis.*