Techniques for Generating Communicating Systems[1]

James M. Neighbors
Bayfront Technologies, Inc.[2]
James.Neighbors@BayfrontTechnologies.com

Abstract: This position paper presents a simple triad model for generating communicating systems. We show a simple architectural and instance example of the model as web services. This example demonstrates the inherent danger presented by detail, version and configuration in attempting to maintain a model of concrete implementation instances by the generator. We point out that in the past generators have solved this problem by forming private simple abstractions of implementation instances. Currently meta-definitional forms such as XML are enjoying resurgence. Because of market forces these meta forms have complex version and configuration spaces. We have found that the old technique of forming private simple abstractions of meta forms still works well to isolate the details, version, and configurations of instance meta forms such as XML. Finally we discuss our belief that the formation of these private abstractions is the creation of domain hierarchies.

Introduction: Triad Communicating Systems Model

The Analysis Model presented in Figure 1 is a simple *triad communicating systems model* we have found to be useful in developing communicating systems. The triad Analysis Model breaks the parts of a communicating system into dataforms, process, and protocol. The *dataforms* are the set of data produced and consumed by the *process*. The dataforms are also prioritized, wrapped and unwrapped into data packets by the *protocol*. The process and protocol asynchronously message each other with dataforms and protocol events.

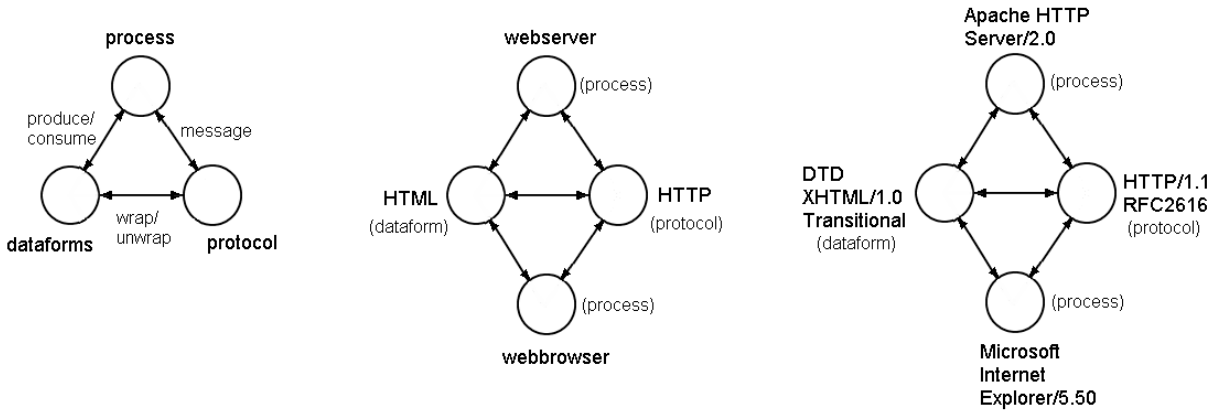


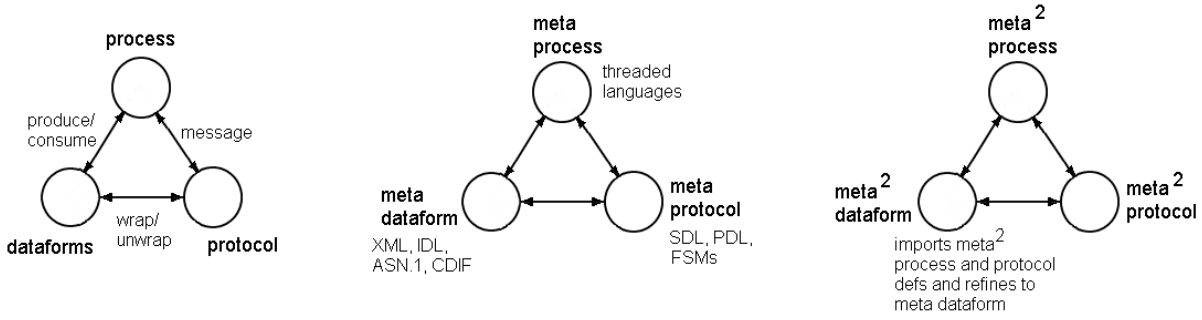
Figure 1: Triad Communicating Systems Model and Examples
(level of abstraction *decreases* left to right)

As an example only the Architectural Model of Figure 1 shows two triads being combined to model the client-server communications relationship of serving web pages. In this case the two triads may be combined into a communicating system because they share a dataform (HTML) and a protocol (HTTP). The example Architectural Model is the level of abstraction maintained by most generators. In other words if we wished to build a generator for web serving and browsing systems, then the Architectural Model is the level of model maintained internal to the generator. The Instance Model of Figure 1 shows why this is so. The Instance Model represents what one real world concrete implementation of this Architectural Model might look like. Each element of the example Instance Model has a very complex detail, configuration, and version space. In addition each element is under the control of a separate social organization (i.e., W3C, Apache, Microsoft, IETF). The Architectural Model maintained by our hypothetical generator can't be expected to maintain the day-to-day changes in a collection of social organizations – many of which might lead to short-term incompatibilities. Classically generators (and people) have resorted to feature subsets that are represented in the simpler Architectural Model maintained by our hypothetical generator. *In essence 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.* The low-level complexity is then isolated to the refinements from the higher level of abstraction. In addition the higher level of abstraction only has to characterize parts of the low-level complexity that it wishes to use to solve its particular set of problems.

[1] This work was supported in part by U.S. National Science Foundation grant DMI-9960830.
[2] Author may be reached at Bayfront Technologies, Inc., B9-231, 1280 Bison, Newport Beach, CA 92660, USA.

Meta Models

Many groups have desired to uncouple the definition of data produced and consumed by a process from the implementation of that process. The desire arises from the untenable position discussed in the previous section of attempting to maintain data definitions or processes or protocols in a complex version and configuration space. It arises in all problem domains in computing. It is particularly acute in communicating systems. The solution to this data problem has been to define data separate from process as our triad model does. *Meta data models* have been used to define data in this way. A meta data model is a description (possibly self descriptive) that characterizes the legal dataforms for a class of processes. XML is currently the best known meta data model language [XML] but others such as Common Data Interface Format (CDIF) [CDIF], Interface Definition Language (IDL) [S89], and Abstract Syntax Notation (ASN.1) [ASN] have existed for years.



Analysis Model | Meta Model | Meta-Meta Model

Figure 2: Meta-Model Hierarchy
(level of abstraction *increases* left to right)

In Figure 2 we show the evolution of our simple triad Analysis Model into the first level Meta Model. Once again realize 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.* XML is an example of this defense related to dataforms. The Meta Model shown in Figure 2 also naturally extends this defense to the protocol and process elements. Meta protocol languages do exist. An example is Specification and Description Language (SDL) mostly used to define telephony protocols [SDL]. Bayfront Technologies Computer-Aided Protocol Engineering (CAPE) product supports Protocol Description Language (PDL). Most protocol description schemes are based on finite state machines (FSMs). As with the Instance Model of Figure 1 market forces have recognized the power of meta dataforms and they struggle for control of them. The result once again is related families of meta dataforms (e.g., XML) with complex detail, version, and configuration spaces. Once again we have no choice but to erect the classical defense by establishing the Meta-Meta Model of Figure 2. The meta-meta dataform must abstract the core elements of a family of meta dataforms such as XML showing and refining only the common definitions we wish to use in our highest level triad model. An alternative approach we *do not* advocate is currently underway where each of the meta dataform instances attempts to define all others.

Conclusions

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 [N98]. Notice that each of these levels has a domain-specific language, optimizing transformations, and refinements to other domains [N89]. 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 [N96].

References

[ASN] ITU Abstract Syntax Notation 1, http://www.itu.int/ITU-T/asn1/ or http://www.asn1.org/resources.htm

[CDIF] ISO Software & System Standards Common Data Interchange Format ISO/IEC JTC1/SC7/WG11 http://www.iso.ch/

[N96] Neighbors, James M., "The Benefits of Generators for Reuse", Panel: *On the Future of Generators*, Baxter, I., moderator **Fourth International Conference on Software Reuse**, Sitaraman, M., editor, pp. 217, April 1996, Orlando, Florida, IEEE Press, 1996. http://www.bayfronttechnologies.com/l02draco.htm#icsr4g

[N89] Neighbors, James M., Chapter 12: "Draco: A Method for Engineering Reusable Software Systems", in **Software Reusability**, *Volume I: Concepts and Models*, Ted Biggerstaff and Alan Perlis, Editors, ACM Frontier Series, Addison-Wesley, 1989. http://www.bayfronttechnologies.com/l02draco.htm#awchap87

[N98] Neighbors, James M., "Domain Analysis and Generative Implementation", Panel: *Linking Domain Analysis and Domain Implementation*, Frakes, W., moderator **Fifth International Conference on Software Reuse**, Devanbu, P., and Poulin, J., editors, pp. 356-357, June 1998, Victoria, Canada, IEEE Press, 1998. http://www.bayfronttechnologies.com/l02draco.htm#icsr5g

[SDL] ITU Specification and Description Language Standard Z.100, http://www.itu.int/ITU-T/studygroups/com17/sg17-q13.html or http://www.sdl-forum.org/Publications/Standards.htm

[S89] Snodgrass, R., *The Interface Description Language*, Computer Science Press, 1989.

[XML] W3C, *Extensible Markup Language (XML) 1.0* (Second Edition), W3C Recommendation 6, October 2000. http://www.w3.org/TR/REC-xml