Application Integration:
Are We Going to Take It Seriously?

Position paper submitted to the Berkeley-Stanford CE&M Workshop:
Defining a Research Agenda for AEC Process/Product Development
in 2000 and Beyond
(August 1999)

Ulrich Flemming
Professor of Architecture
School of Architecture
and Institute for Complex Engineered Systems (ICES)
Carnegie Mellon University
Pittsburgh, PA 15213
ujf@cmu.edu

James Snyder
Senior Research Scientist
Advanced Technology Laboratory
Lockheed Martin Corporation
Camden, NJ 08102
jsnyder@atl.lmco.com

1 Scenario

Building design is a multi-team effort, and the teams participating in specific projects increasingly base their work on computer support, which has the potential to support also collaboration and information exchange between teams. In one possible scenario, different teams collaborate synchronously by working on the same data model or 'document' over some network connection using networked 'groupware' or shared 'electronic whiteboards'. In another scenario, teams within a large integrated firm, which combines all participating disciplines, use software written against a shared data model so that information, even proprietary data, can be freely exchanged. The cost of developing such a shared data model and custom software may be feasible because it is used and re-used for years to come. These scenarios have in common that they work in practice because they are supported by commercially available or proprietary software.

But as Eastman has pointed out [Eastman 1999], there exist realistic collaboration scenarios in the building industry that differ significantly from the above ones, and they may indeed be the ones of greater practical significance, given the fragmented structure of the industry. In these scenarios, the firms or design teams collaborating on a building project join work *only for this specific project*. They use local software to the degree that is appropriate to them; we call any such local software an *application* in the following. An application uses algorithms and internal representations/data models tailored to the particular design task it supports. Each of these local models captures a particular universe of discourse that differs - to some degree at least - from those of the other teams. Each team uses local persistent data storage facilities, i. e. a local database, and it does this again to the degree it deems necessary and appropriate. It is likely that the individual applications were developed independently of each other and use different programming languages; that is, they are *heterogeneous*. It is furthermore to be expected that different teams enter the project at different times, which are, in addition, unpredictable at the outset. The length of time a team spends on a project may also vary significantly; for example, an energy consultant may be used only during a short period in the early conceptual phase, while an interior design team may join the project late, but remain involved until occupation starts and beyond.

On the other hand, many of the decisions made by one team must be communicated to some of the other teams because they have ramifications for the work done by those teams. Each team intends to take advantage of the fact that it has a computable design representation and to use that representation to automate information exchange to the largest possible degree. This is the scenario addressed in the present paper.

It has been observed that the approaches suggested for a more asynchronous information exchange typically fall into two categories ([Rolland and Cauvet 1992] pages 39 and 40): (1) The *fully integrated approach*, which relies on a unique, shared representational schema to which all subsystems have to conform; and (2) the *federated approach*, in which each local system has its own schema and is responsible for controlling its interactions with other systems by deciding which information should be in- and exported. We agree with Eastman and other observers who conclude that neither approach is capable of supporting the scenario with which we are concerned. We also agree the present - practically very realistic - scenario raises theoretical and technical problems that have not been solved. That is, these problems represent a major stumbling block for application integration in the building industry as envisaged in our scenario.

We lay out in section 2 the requirements that need to be met if this type of asynchronous collaboration between heterogeneous, distributed applications is to be supported by software. Section 2 is highly influenced by our own experiments with an asynchronous application integration environment [Snyder and Flemming 1999]. These experiments were conducted - in part - in the context of the Agent Communication Language (ACL) project, in which both authors participated [Khedro 1995]. The ACL project, to the best of our knowledge, is the only research project to try the federated approach in the context of building design.

This background gives our paper a distinctly concrete, hands-on flavor, which we expect to differ from some other submissions that may paint with a broader brush. But we submit that the experience reported and issues raised have wide-ranging ramifications for research on collaborative design environments supporting our scenarios and that they are practically significant to the degree that these scenarios are. We derive in section 3 a research agenda to address the issues raised by the requirements in section 2.

2 Requirements

The general approach that suggests itself for the type of application integration required by our scenario is some form of mediated architecture [Wiederholt 1995]. We outline below some main issues that have too be resolved in a mediated integration environment that supports this scenario.

Schema Mapping

We cannot expect that the internal models used by the different applications are conceptually equivalent. Information exchange requires mappings between the different conceptual schemas involved and may include synthesis and re-synthesis of concepts. Note that this type of communication goes beyond the kind of type conversion called *transduction* in [Eastman 1994].

Semantic correctness must be maintained in both directions. Since applications do not export every piece of information they generate (some of which may be proprietary anyway), information loss will occur during the exchange; but this loss has to be carefully controlled. On the other hand, some information is only relevant if it is delivered in a timely manner, while other information is relatively static. The information exchange has to be able to take such varying rates into account. This makes, for example, any exchange where the unit of exchange is an entire file inappropriate for our scenario.

Translation Code Generation

We need software to map between schemas or to translate between different representations. There is mounting evidence that this software cannot be written reliably by hand. Both our own experiments as well as reports found in the literature attest to this [Dubois et al. 1995]. This type of code is of stultifying repetitiveness: assignment statement after assignment statement, each preceded by layered conditionals which ascertain that a value has been in fact been defined, that it is of the correct type, within the correct bounds etc., and this may have to proceed up an inheritance structure of arbitrary length. The copy-and-paste style of programming that this leads to is notorious for leading to residual bugs, which must be found, object-by-object, link-by-link and attribute-by-attribute. It is practically infeasible to engage in this type of effort if the code is to be used for collaboration in only one project and possibly a very short duration, as our scenario suggest.

We conclude from this that the generation of translation code must be automated to the greatest possible degree from high-level specifications that are short enough to be written by hand. Our own experiments demonstrate the feasibility of this approach [Snyder and Flemming 1999]. They demonstrate especially that compilation represents a form of formal the correctness check: translation code is correct *if it compiles* to the degree that the specifications are correct. To give a concrete example of the efficiency that can be gained in this way: in our experiments, we were able to automatically generate 17,000 lines of translation code (as long as the application itself!) from 500 lines of specifications.

Schema Specification Language

We specifically demonstrated that mapping specifications can be written effectively against a schema expressed in a *modeling language* that is rich enough to capture all data that must be exchanged. The schema itself captures the pieces of information that are of mutual interest to the participating applications. It is *shared* in this sense and acts as 'lingua franca': each application commits to translating its sharable data into this schema and external data of interest from the schema, using automatically compiled translation code. The schema must allow for evolution because the cast of participating applications is ever changing in our scenario. Again, we cannot update manually translation code for each application after each schema change.

Language Features

We list below some important requirements a schema specification or modeling language as conceived above has to meet

- **Behavior Modeling**. A schema is in all likelihood object-based, given the prevalence of object-based representations in research and increasingly in practice. We learned that the schema must be able to *model the behavior of object configurations* under various update events. For example, if a new version of a solution is exported, where the solution consist of various parts, the parts have to be created as new instances and linked appropriately, whereas links to Functional Units expressing requirements may have to point to already existing objects. Conversely, if a completely new object configuration is to be created, *all* instances in the configuration, including all links, have to be newly created. This type of relationship management goes beyond the traditional distinction between deep and shallow copies; it requires the schema to be able to manage relationships at a finer granularity and in response to specific update events. Existing programming and information exchange languages are not rich enough to capture these behaviors; we need a true modeling language.
- **Multiple classifications**. The language must support multiple classifications of objects and object configurations. Moreover, it should not use multiple inheritance for this purpose.
- **Constraints**. The language must support some form of constraint specification. The literature is sometimes confused on this issue. To clarify this, it is useful to distinguish well-formedness of data from the well-formedness of a design model. A constraint assuring the former may, for example, restrict the values allowed for an attribute representing geographical latitude to the allowed angle; a constraint assuring the latter may for example prevent a second floor from being described before the supporting first floor has been modeled. A schema specification language should definitely

provide for constraints (or axioms) assuring well-formedness of data being exchanged. One should extremely cautious about constraints dealing with model well-formedness. Many workflows and design explorations would be unduly restricted by ill-conceived constraints on the model itself.

Software Architecture

Modeling languages and mapping specifications should not be written in a vacuum. They should be conceived against an anticipated software architecture; for example, it makes a big difference if we know that this server will have its own run-time memory to store temporary versions of object configurations or not. Any reliable relationship management may in fact require this.

Platform Independence

On a more technical level, any application integration environment conceived along these lines should no restrict participating applications in terms of their native programming language, data model, local persistent storage facilities, hardware etc.

3 Research Agenda

To create an application integration framework able to fully and effectively support the asynchronous collaboration of heterogeneous applications as envisaged in our scenario, the following issues must be addressed:

1. **Modelling language**: The features and expressive power needed by a schema specification language able to model object behaviors including relationship management, to serve as lingua franca supporting schema mappings between applications, and to support translation code generation.
2. **Mapping specifications**: The syntax and semantics of mapping specifications written against a shared schema specified in a modeling language, on the hand, and a client (native) schema or data model, on the other hand.
3. **Software architecture**: An effective form for a mediated architecture against which schema specifications can be written so that substantial portions of this architecture can be automatically generated.
4. **Experiments** with a functioning implementation. The effectiveness of 1-3 can only be tested by running realistic experiments with a suitably robust implementation. Many hard issues, as our own experiments suggest, are discovered only in this way.

References

[Dubois et al. 1995] Dubois, A. M., Flynn, J., Verheof, M. H. G., and G. L. M. Augenbroe: “Conceptual Modelling Approaches in the COMBINE Project” in *Proceedings of ECPPM ‘94 - The First European Conference on Product and Process Modelling in the Building Industry* R. J. Scherer (ed.), Dresden, Germany, October 5-7, 1995, Rotterdam: A. A. Balkema.

[Eastman 1994]Eastman, C.M. : “A Data Model for Design Knowledge” in *Knowledge-Based Computer-Aided Architectural Design* G. Carrera and Y. E. Kalay (eds.), Elsevier Science B.V., Amsterdam, The Netherlands, 1994, pp.95-122.

[Eastman 1999] Eastman, C. M. “Information exchange architectures for building models’ *Proc. 8th International Conference on Durability of Building Materials and Components*, Vancouver, Canada, May-June 1999

[Khedro 1995] Khedro, T., Case, M. P., Flemming, U., Genesereth, M. R., Logcher, R., Pedersen, C., Snyder, J.,Sriram, R. D. and P. M. Teicholz: “Development of a Multi-Institutional Testbed for Collaborative Facility Engineering Infrastructure” in J. P. Mohsen (Ed.) *Computing in Civil Engineering*: *volume 2: Proceedings of the Second Congress held in conjunction with the A/E/C Systems ‘95*, Atlanta, GA, June 5-8, New York: American Society of Civil Engineers, 1995, pp. 1308-1315.

[Rolland and Cauvet 1992] Rolland, C. and C. Cauvet: “Trends and Perspectives in Conceptual Modeling” in

Conceptual Modeling, Databases, and CASE: An Integrated View of Information System Development P. Loucopoulos and R. Zicari (eds). New York: John Wiley & Sons, Inc., 1992, pp. 27-48.

[Snyder and Flemming 1999] Snyder, J. and U. Flemming. “Information sharing in building design” *Proc. CAADFutures ‘99*, Atlanta, GA (G. Augenbroe and C. Eastman, ed.s) Kluwer, New York, 1999

[Wiederhold 1995] Wiederhold, G.: “Mediation in Information Systems” *ACM Computing Surveys*, **27**(2), New York: ACM Press, 1995, pp. 265-267.

[