Cross-aspect Queries and Dynamic Views for Model-based Development*

David Oglesby, Kirk Schloegel, Eric Engstrom

Aerospace Electronic Systems Research Lab, Honeywell International
3660 Technology Drive, Minneapolis, MN 55418
{david.oglesby, kirk.schloegel, eric.engstrom}@honeywell.com

Abstract

Some important characteristics of model-based development are the ability to see more of a design at a time (at a higher level of abstraction), and to see its interactions more directly than ASCII text can support. This works well until the design scales so large that the model becomes overwhelming. To address large designs, models are often broken up into systems of interacting models. However this can obscure important interactions. A cross-model query and viewing capability makes it possible to specify the components and interactions that are of interest so that cross-aspect and cross-model interactions can be seen in a single view without excess clutter.

1. Introduction

A key objective of domain-specific modeling is to represent a design in a more intuitive manner than textual programming languages and to show relationships between components. In large part because graphical representations can be more intuitive compared to text, they can be more concise, showing us more of a design. Also connections can be represented as arcs between components; and so, relationships among various components become immediately clear. This is something that cannot be accomplished with text.

Due to these and other benefits, model-based design is being applied to increasingly large systems. Scalability then becomes a critical property of domain-specific modeling (DSM) tools. The first such hurdle that occurs with only tens or hundreds of modeling entities is the problem of partitioning the design into multiple logical sheets. The assumption is that a large design partitioned into a number of sheets is easier to comprehend than one that is viewed as a single model. Typically, this partitioning is performed by the user and is based upon primary functionality.

This approach detracts from the original draw of model-based design, however, because components that interact in important ways may appear in different views. Current DSM tools provide little support for visualization based upon alternative partitionings. Hence, the so-called *tyranny of the dominant decomposition* [4] is the result.

Additionally, as systems become more complex, it is also desirable to capture multiple aspects of system designs. For example, an embedded system design should represent the

* This material is based upon work supported by the United States Air Force under Contract No. F33615-00-C-1705. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

hardware as well as the software and connect the two. These aspects are often represented with different modeling notations.

2. Dynamic Views

We believe that users should be free to customize temporary views in order to visualize designs based upon arbitrary partitionings. These views need not (and indeed, should not) be maintained. Instead they provide a dynamic snapshot of the aspects of interest. As such, they should be easy to create and recreate. In these views the user should be able to interact with the model as in the standard persistent views. That is properties can be changed and components can be added or removed. They should also be domain independent, so cross-aspect interactions become viewable. We refer to these as *dynamic views*.

Figure 1 through Figure 4 illustrate this approach on a component interaction model of a simple flight control system. Figure 1 shows a system model.

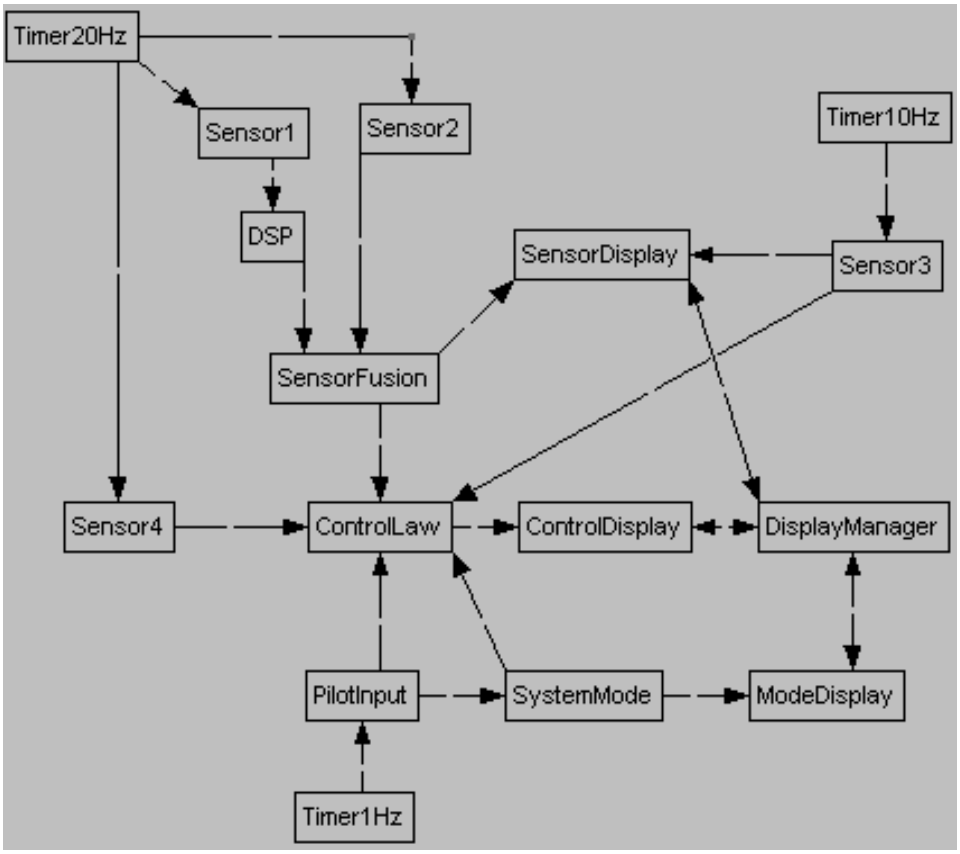

Figure 1 A component interaction model of a simple flight control system

A typical partitioning of this design might break the model into two sheets. One of these would show the sensor, pilot input, and control law components, while the other would show the display components. (See Figure 2 and Figure 3.[1])

[1] Note that a number of components (SensorFusion, Sensor3, SystemMode, ControlLaw) appear on both sheets in order to anchor arcs to other components. These duplicated components represent a single logical component.

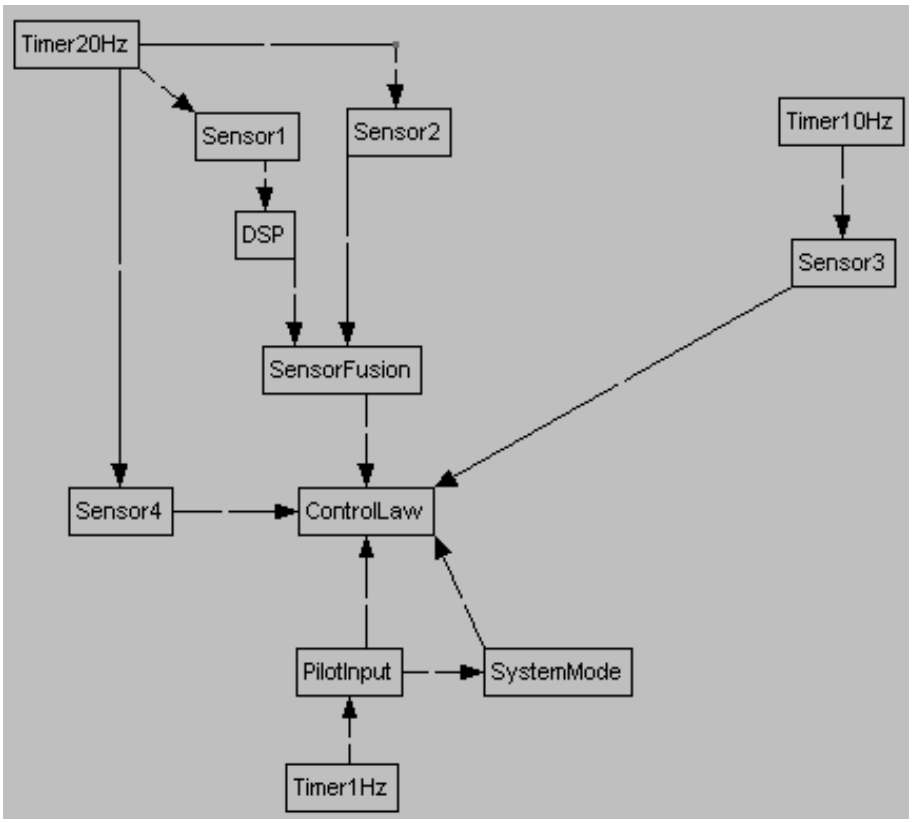

Figure 2 A sheet from the model in Figure 1 that has been partitioned with respect to control functionality

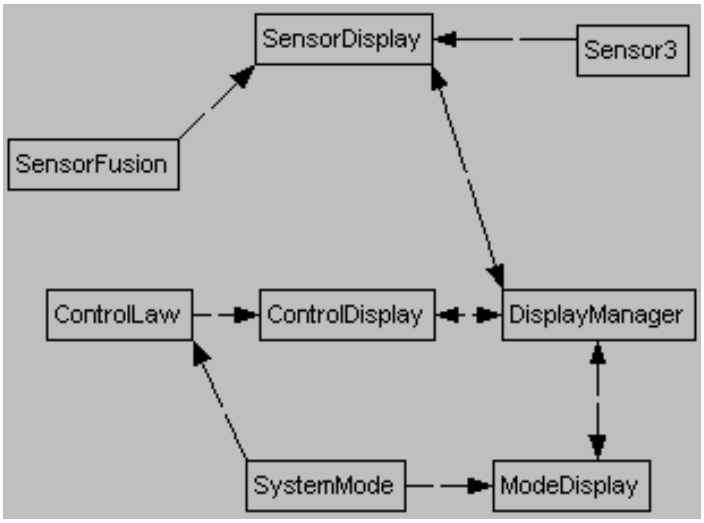

Figure 3 A sheet from the model in Figure 1 that has been partitioned with respect to display functionality

In the development of large systems, multiple engineers collaborate. Each engineer might be focused on a different aspect of the system. Hence, there is some kind of partitioning of the work effort. This partitioning may not always conform to the partitioning of the design into multiple sheets. For example, a sensor stream engineer might be interested in a view that shows the interaction slice from timers to sensors to displays. However, under the partitioning shown in Figure 2 and Figure 3, this slice does not exist in a single view. Instead, it will be necessary to jog between views to visualize the interactions. Essentially,

the benefits of partitioning the design are lost to the sensor stream engineer. This view is shown in Figure 4.

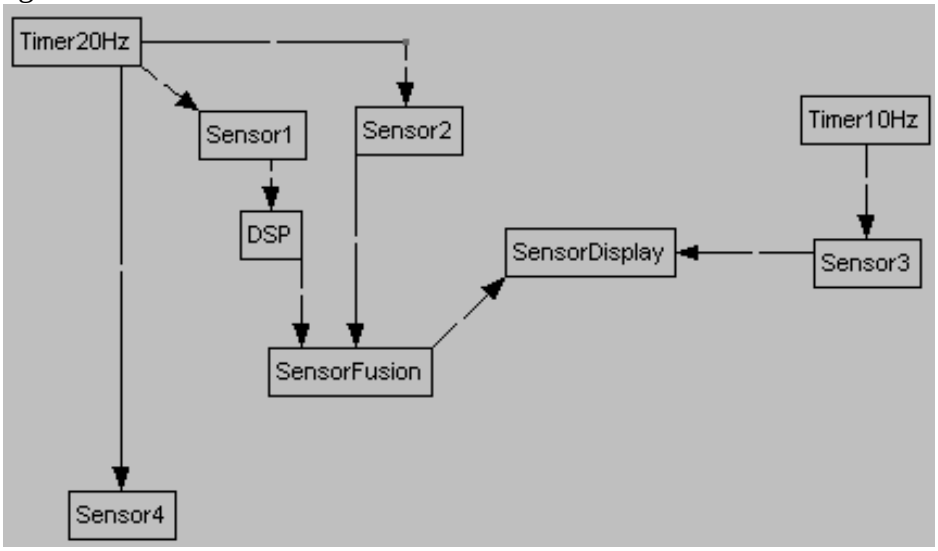

Figure 4 A view of the model from Figure 1 that involves the timer to sensor to display interaction slices.

Cross-aspect views are even more important in development environments that are composed of multiple modeling notations or tools. Here, integrated modeling notations implemented in one or more DSM tools are used to cooperatively specify the various aspects of a complex design [1][2][3]. Such integration enables the specification of designs as systems of disparate, yet interacting models. Current DSM tools cannot in general visualize entities from other modeling notations. This capability is useful for designs involving cross-aspect interactions.

Suppose the component interaction model from the example above was linked to a particular hardware model such that components (from the component interaction model) could be allocated to processors (from the hardware model). Such an allocation might be computed automatically for a subset of the components based upon certain criteria. The remainder of the components would be allocated by hand. At such a time, it would be useful for the user to be able see a dynamic view of the state of the system of models that includes all unallocated components and their interactions along with all processors with available computation and memory resources. Figure 5 illustrates such a view.

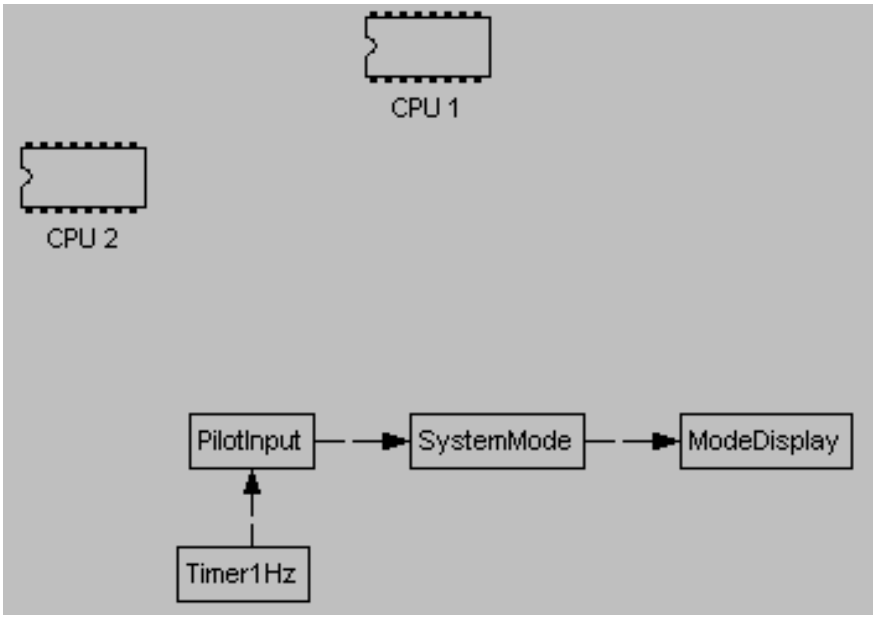

Figure 5 A portion of the model from Figure 1 along with two modeling entities that represent CPUs from a hardware model

A related cross-aspect viewing problem exists in the context of heterogeneous hierarchical model composition. Hierarchical composition of models is a commonly used mechanism for reducing model complexity. Under this framework, modeling entities may encapsulate one or more subdiagrams that describe internal details. Typically, modeling entities encapsulate subdiagrams that conform to the same modeling notation as the parent entity. By this mechanism, hierarchical model composition is supported. However, it is also useful to allow subdiagrams to describe different aspects from their parents. (That is, the parent and child modeling notations are not the same.) An example is hybrid systems, in which each state of a state-transition diagram contains a subdiagram describing the continuous behavior that is associated with it [5][6][7][8]. The problem here is that there is no capability to view flattened designs that span modeling notations.

Figure 6 and Figure 7 show an example. Figure 6 models the behavior of the SensorFusion component from Figure 1. In particular, after the Sensor2 and DSP components make their data available, the SensorFusion component runs a sensor fusion algorithm. Then it makes the fused data available to the ControlLaw component. Figure 7 shows a merged view of a portion of the component interaction model along with the internal behavioral model of the SensorFusion component. This unique view is enabled by the domain independent nature of dynamic views.

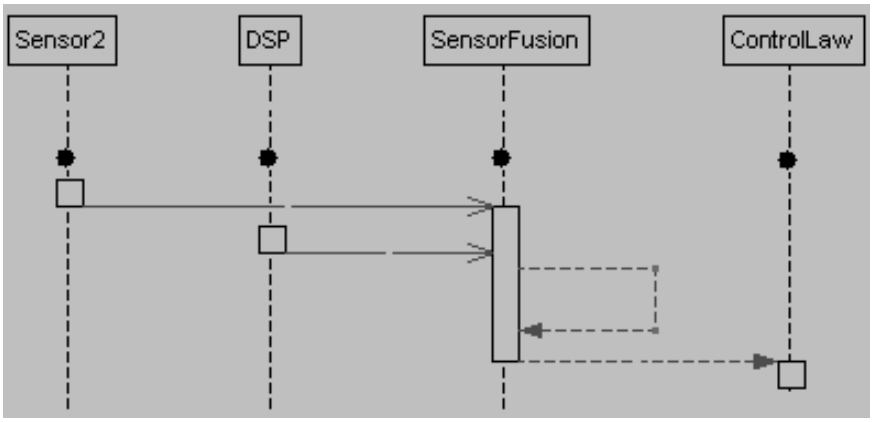

Figure 6 A model of the internal behavior of the SensorFusion component

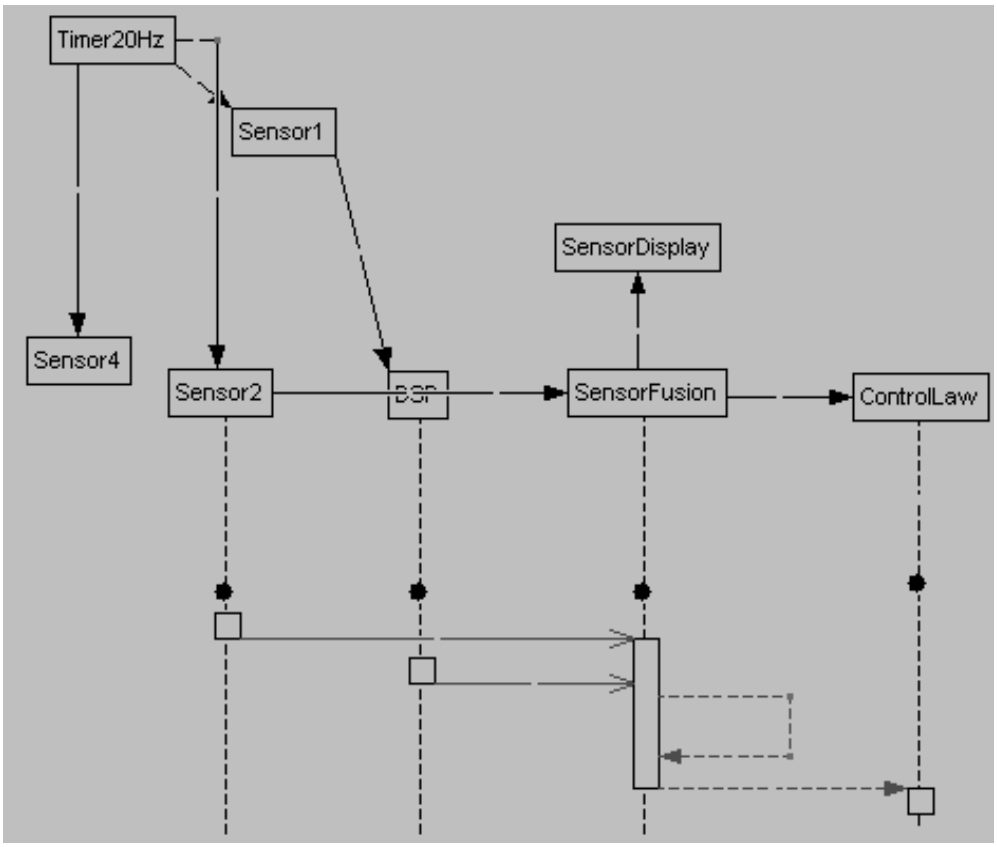

Figure 7 A cross-aspect dynamic view of a portion of the component interaction model along with the internal behavioral details of the SensorFusion component.

3. Query Model

One way to specify dynamic views is with a model query. As we are working in visual domains, a visual query language seems appropriate [9]. We use a domain-independent query language; with it a user can create queries for designs across modeling notations. The user specifies the properties and connections of interest in a query model and a dynamic view of the result is automatically generated. Since these queries are models they can be saved and reused; thereby supporting the easy recreation of dynamic views.

Figure 8 shows the query model that results in the dynamic view from Figure 7. The dynamic view will contain all items with names beginning with "Timer" or "Sensor", all items named "SensorFusion" and its subdiagram components, arcs connecting timers to sensors, objects to which SensorFusion is connected, and anything between a sensor and SensorFusion[2].

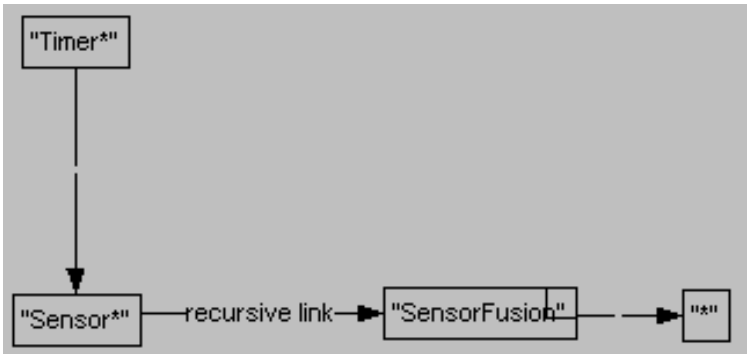

Figure 8 The query model that results in the dynamic view from Figure 7

In this example, queries are based on pattern matches with object names and connectivity. Sets of components with matching names and connections will be returned. Additionally a user can specify object type or property values or ranges of values. Nodes that appear in multiple views (e.g. ControlLaw) will appear only once in the resulting graph; however, if two distinct nodes match a query, both will be returned.

4. Conclusions

As DSM technologies increase in popularity, they are being applied to more complex designs at greater levels of detail. These larger models can be used for additional analyses and more complete automatic code and test generation. Unfortunately, they also lose the comprehensibility that was initially a key attraction. Domain-independent dynamic views help to improve scalability by addressing this problem. By querying for relevant portions of a system of models, a user can interact with all of the components of interest to him or her and only those components.

Dynamic views are domain-independent to support views across modeling notations. As a result, working in these views lacks some of the benefits of domain specific modeling. Among the considerations is constraint checking. If components are added or removed and properties set in a dynamic view, the constraints will have to be checked when the changes are committed to the model. Taking this further, it may be possible to reference the relevant metamodels when working in a dynamic view to enforce such constraints and derive other domain-specific benefits. The work on metamodel composition by Ledeczi et al. [10] and aspect-oriented metamodeling by Clark et al. [13] might be relevant here.

Our current query modeling notation is preliminary and still evolving. Furthermore, the query engine behind this notation is still rather ad hoc in nature. It has not been optimized or analyzed for efficiency and correctness. Of particular interest is how this query mechanism deals with cycles and recursive subdiagrams. Further work in this area can be guided by previous graphical query research [9][11]. Additionally, the OMG is working toward

[2] The recursive link will result in all arcs and nodes that form a path between its endpoints. In this example these consist of the following: the arc from Sensor1 to DSP, DSP, the arc from DSP to SensorFusion, and the arc from Sensor2 to SensorFusion.

Query/View/Transformation standard that may prove relevant. An initial open-source implementation of dynamic views and query models can be found as components of the DOME metamodeling tool suite [12] at http://www.htc.honeywell.com/dome.

5. References

[1] A. Bakshi, V.K. Prasanna, and A. Ledeczi. *MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems,* In Proc. of Workshop on Languages, Compilers, and Tools for Embedded Systems, 2001.

[2] Honeywell International, *The Systems and Applications Genesis Environment (SAGE),* htttp://www.honeywell.com/SAGE.

[3] K. Schloegel, D. Oglesby, E. Engstrom, D. Bhatt. *A New Approach to Capture Multi-model Interactions in Support of Cross-domain Analyses*, 2001.

[4] P. Tarr, H. Ossher, W. Harrison and S.M. Sutton, Jr. *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. In Proc. of the International Conference on Software Engineering (ICSE'99), 1999.

[5] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3-34, 1995.

[6] R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III: Verification and Control*. LNCS 1066. Springer-Verlag, 1996. Proceedings of the Third International Workshop.

[7] N.Lynch and B.H. Krogh, editors. *Hybrid Systems: Computation and Control*. LNCS 1790. Springer, 2000.

[8] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 447-484. Springer-Verlag, 1991.

[9] W. Ni, T.W. Ling. *GLASS: A Graphical Qeuery Language for Semi-Structured Data,* In Proc. of Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03), March 2003.

[10] A. Ledeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti, “On Metamodel Composition”, *Proceedings of IEEE CCA 2001*, 2001.

[11] I.F. Cruz, A.O. Mendelzon, P.T. Wood, "A Graphical Query Language Supporting Recursion", *SIGMOD Record Vol 16, Issue 3*, 1987.

[12] DOME is an open source research project available at http://www.htc.honeywell.com/dome

[13] T. Clark, A. Evans, S. Kent, "Aspect-OrientedMetamodelling", http://www.cs.kent.ac.uk/people/staff/sjhk/publications/CJ03/content.pdf. 2003.