Peridigm Summary Report: Lessons Learned in Development with *Agile Components*

David Littlewood, John Mitchell, and Michael Parks

May 27, 2011

Executive Summary

1 What is Peridigm?

Brief overview of peridynamics and discretization.

Summary of Peridigm capabilities.

2 What is Agile Components?

2.1 The Agile Components Interface Paradigm: The ModelEvaluator Class

ModelEvaluator as a manifestation of the parameters-in responses-out paradigm.

2.2 *SimpleODE* as an Agile Components MiniApp

Overview of SimpleODE, core components, and control flow of code as instantiation of Agile Components methodology.

3 Lessons Learned

3.1 Where the Agile Components Methodology Works Well: Target Use Cases

3.2 Shortcomings of the Agile Components Implementation

3.2.1 Handling Model State

History Dependance in Material Models Compare/contrast simple PDEs with material models with state (e.g., damage, plasticity, etc.) Explain observer concept as mechanism to update state. Alternative approach: mechanism to hand in and out state data at each ME call, rather than have ME store state data that is updated by observer. However, in SimpleODE control flow outlined above, the time integrator is generic and does not know to pass in/out state data to the ME for the material model. So, where should state data live?

Load Rebalance and Mesh Adaptivity Discuss notion that maps (e.g., data distribution) are state data contained within ME class. ME class is thus not referentially transparent. Options to resolve issue are (1) rewrite all ME classes to accept maps as input, as opposed to current ME methodology where ME generates maps and initial conditions / initial data. This represents a massive recoding effort. (2) Generate remap classes for each model evaluator to remap ME from old map assignment to new map assignment. Due to internal data stored in ME, remap class is specific to each ME class (time integrator, material model, etc.) This likely also represents massive coding effort.

Data Distribution ME enforces use of maps, not blockmaps. Mapping multiple-dof-per-node data to a map from the more natural blockmap is cumbersome.

3.2.2 Time Integration

Discuss lack of availability of second-order integrators. These are now available via PIRO. Discuss Rythmos assumptions on model evaluator; lack of ability to return control to main() between timesteps (needed for iterative contact algorithms, etc.); issues having to stack vectors etc. Discussions with Andy lead to revision of Peridigm control flow from that used by SimpleODE to a three-object model: a controlling object, a time integrator, and a material model wrapped in a ModelEvaluator.

4 Recommendations

Despite shortcomings in the current implementation, we are supportive of the goals of *Agile Components*. For its target use cases, the Agile Components methodology has proven to facilitate rapid development of powerful application code. For applications outside the target use cases, The potential rapid development capabilities that agile components may provide are simply too tantalizing to overlook. To that end, we make the following recommendations, which we feel would improve the impact of continued agile components efforts to new applications.

I. Improved documentation along with easy-to-understand demonstration code, such as SimpleODE. This will greatly shorten the learning curve for new users, and help them to understand how their application fits within the agile components methodology. SimpleODE may be useful as a starting template for new users to develop their own application based upon Agile Components.

II. Item 2.

III. Item 3.