IOWA STATE UNIVERSITY
Department of Computer Science

Laboratory for Software Design
http://design.cs.iastate.edu

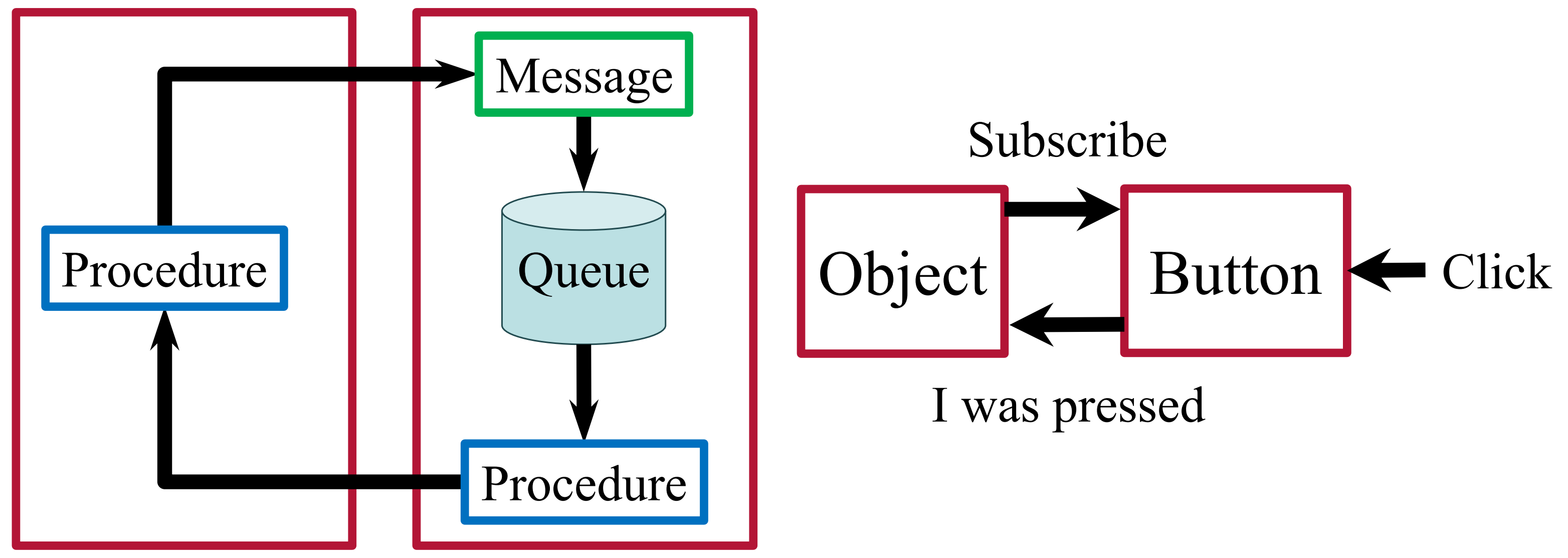
Presenter: Jackson Maddox (jlmaddox@iastate.edu) Advisor: Hridesh Rajan (hridesh@iastate.edu)

Integrating Event-Driven and Capsule-Oriented Programming

Introduction

Problem
Panini is a capsule-oriented programming model that helps developers take advantage of concurrency [3]. However, it is difficult for developers to use Panini when developing applications with common event-centric components such as user interfaces. There is no way for a capsule to ask another to notify it when something happens, such as a button click.

Solution
An integration of Panini and events would allow developers to more easily design concurrent systems that have event-centric components.



Differences in interaction between capsules (left) and with events (right)

Objectives

This project combines the Panini and event-driven programming models into a single, integrated model respecting Panini's properties that make it easy and safe to use.

Methods

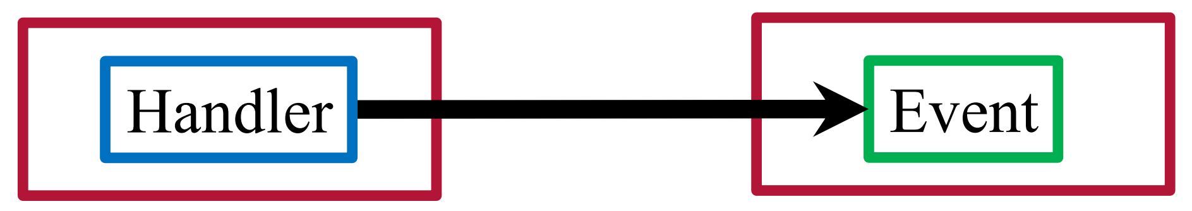
- Implemented event-driven capabilities into @Paninij, an implementation of the Panini model.
- Code is available at: https://github.com/jlmaddox/panini/tree/event
- Examined current literature to see how others have handled the challenges faced during this project

Results

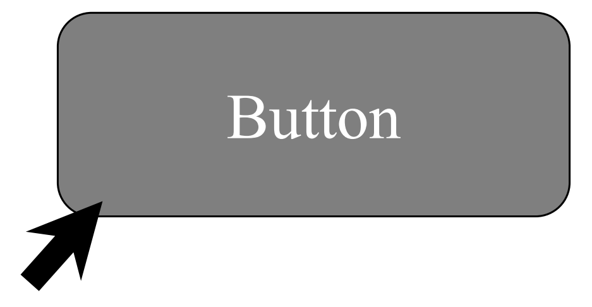
The result is a version of @Paninij with event-driven capabilities.

Capsule Events
- Capsules can contain events, such as a button click
- Capsules can specify some code that handles events occurring

The Model
This is how the integrated model works:

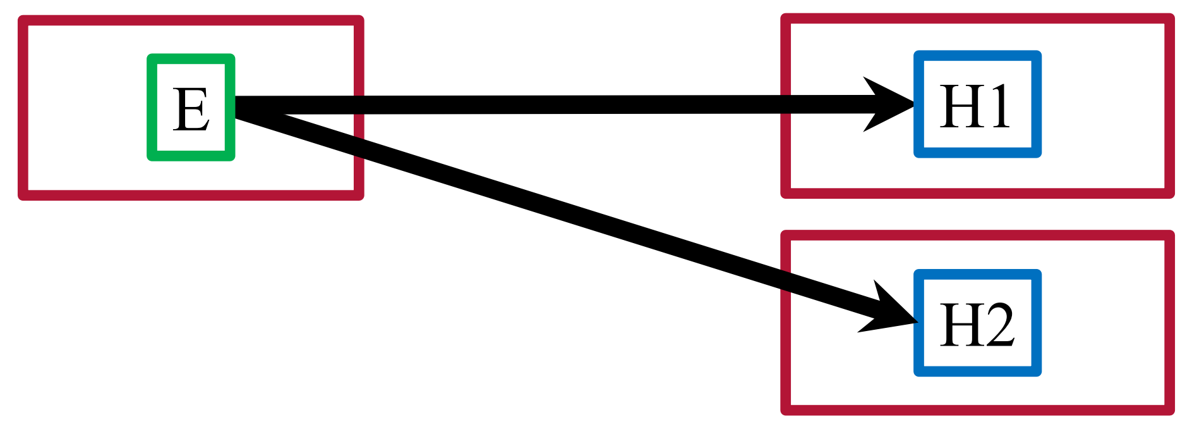


Step 1: At startup, capsules subscribe to the events it is interested in. Multiple capsules can subscribe to a single event.

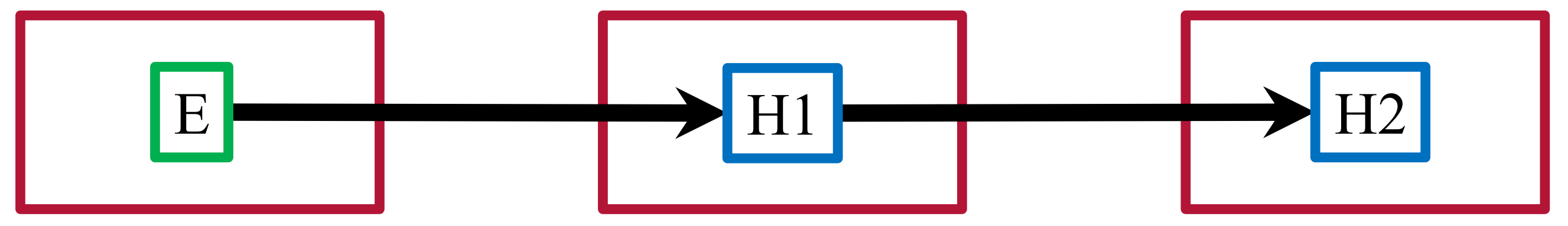


Step 2: The event occurs

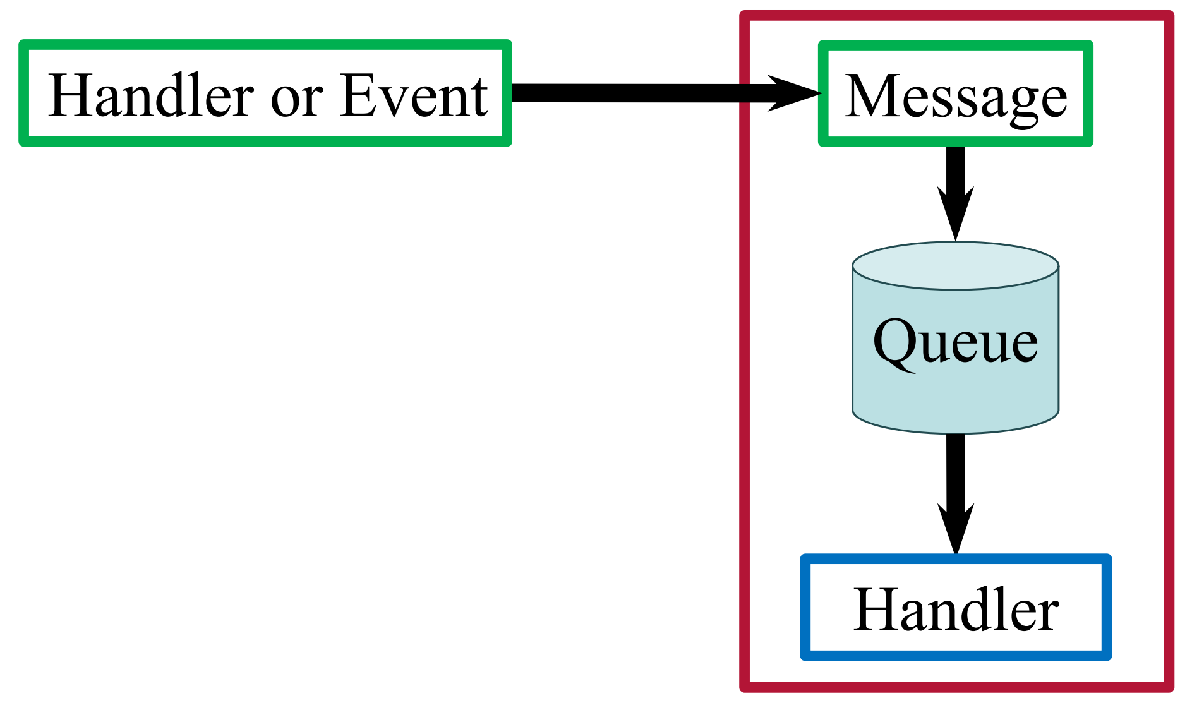
Step 3: One of two things happens, depending on the developer's choice.



Step 3a: Broadcast notification: E is announced and handlers H1 and H2 only read the message. Send the same message to both simultaneously.



Step 3b: Chain notification: E is announced and H1 or H2 may modify the message. Send the message to one and have it pass it on.



Step 4: Notification message arrives at the subscribing capsule

Challenges
- Designing the notification scheme (step 3) to be performant and thread safe
- Deciding how to handle missed event announcements due to capsules with events starting before their subscribers is an unresolved challenge

Conclusion

This honors research project has led to the following findings:
- It is possible to integrate event-driven and capsule-oriented models into a single model to take advantage of both
- The missed announcements problem can be addressed reasonably by for example simply assuming that some might be missed or by requiring all capsules to start up before executing any code or handling messages
- The integrated model allows developers to create concurrent, thread-safe software that contain event-centric components in a clean manner

Reference

[1] Hridesh Rajan, "Capsule-oriented Programming ," ICSE'15: The 37th International Conference on Software Engineering, Florence, Italy, May 2015.
[2] Mehdi Bagherzadeh and Hridesh Rajan, "Panini: A Concurrent Programming Model for Solving Pervasive and Oblivious Interference," 14th International Conference on Modularity (MODULARITY'15), Fort Collins, Colorado, USA, March 2015.
[3] Hridesh Rajan, Steven M. Kautz, Eric Lin, Sean L. Mooney, Yuheng Long, and Ganesha Upadhyaya. "Capsule-oriented Programming in the Panini Language", Tech. Report 14-08, Computer Science, Iowa State University, August 5, 2014.
[4] Yuheng Long, and Hridesh Rajan, "A Type-and-Effect System for Asynchronous, Typed Events," 15th International Conference on Modularity (MODULARITY'16), Malaga, Spain, March 2016.
[5] Patrick Eugster and K. R. Jayaram, "EventJava: An Extension of Java for Event Correlation," ECOOP'09: The 23rd European Conference on Object-Oriented Programming, Genova, Italy, July 2009
[6] Douglas C. Schmidt, "Reactor: An object behavioral pattern for concurrent event demultiplexing and dispatching," Pattern languages of program design, 1995.
[7] Krohn, Maxwell N., Eddie Kohler, and M. Frans Kaashoek, "Events Can Make Sense," USENIX'07: USENIX Annual Technical Conference, Santa Clara, CA, USA, June 2007.
[8] David Notkin, David Garlan, William G. Griswold, Kevin Sullivan, "Adding implicit invocation to languages: Three approaches," The 1st JSSST International Symposium, Janazawa, Japan, November 1993.

This work was supported in part by US National Science Foundation (NSF) under grants CCF-08-46059, CCF-11-17937 and CCF-14-23370. The opinions are those of the authors and do not reflect the views of the NSF.