| Chris Cargile | CSCI656 | March 20, 2014 | Ch.9 HW Questions |
|---|---|---|---|

*This homework assignment (plus reading) required 2.5 h. Answering questions from the assigned section required .5 h*

### 9.5.1) What are architectural concerns that can be mapped to implemented systems? What are strategies used to map them?

Aspects of architectural concerns that can be mapped to implemented systems include: components and connectors; interfaces; configurations; design rationale; non-functional properties; and behaviors. Strategies for ensuring a vis-à-vis mapping include enlisting frameworks designed to simplify implementations of architectures according to various styles, using an approach that is more manual or may be partly automatable with tools. Selecting the most suitable framework is a factor influenced by the constraints of the style and its implications on the system compared to the functions required and considerations or constraints the framework dictates. To ensure consistency between them, traceability linage and the model between them can be included and/or the implementation effected through a generative technique.

### 9.5.3) What is an architectural implementation framework? How does it differ from middleware?

An architectural framework is a bridge that connects the concepts in a system's architecture to its implementation. Frameworks assist developers in adhering to the constraints and prescriptions of a style by codifying a style through which a system's design can be implemented. Middleware provides a construct that can be the starting point for building systems but is not a construct which adheres to or prescribes any stylistic ideals.

### 9.6.1) Run the Lunar Lander apps <book code> from Ch.9 and examine how control and data are exchanged through the frameworks.

The 3 classes found in the pipe-and-filters-(P/F) style version of Lunar Lander may be run individually, as they each contain a main(). Analyzing the data flowing in and out can be achieved by examining the classes, individually, in the P/F version. Data input is prefaced by a % and output by a #, as indicated in the book description.

Of the roughly 20 classes found in the C2-style version of Lunar Lander, provided by the authors, data and control flow occur differently from the P/F version. In the C2 version, the thread model is conveyed explicitly vs. using java's inherent means of facilitating execution through threading. Data is transferred using parameterized name-value pairs and call-return methods vs. in the P/F style where data/control transfer occurs over system-level processes and streams associated with each process, which are sequentially chained using the command line interface.

### 9.6.2) Compare and Contrast java.io and java.nio packages vis-à-vis the pipe-and-filter style's rules and structure.

**Figure 1.** Pipe-and-filter Architectural Style Rules

| | Java.io | Java.nio |
|---|---|---|

| | | |
|---|---|---|
| Input may be filtered and output eventually formatted by a final filter | Streams input/output may be buffered, flushed, and/or serialized using the library. | Streams input/output may be buffered, flushed, and/or serialized using the library. Additional support for async communications is provided and functionality achieved through so-called 'channels' and 'selectors' vs. 'pipes' and 'filters.' |

**Figure 2.** Architectural structural comparison among packages.

| | Java.io | Java.nio |
|---|---|---|
| Platform Support | JVM, thus multiple platforms | JVM, thus multiple platforms |
| Fidelity | Assists developers in applying P/F-style constraints but does not enforce them. Stylistic points of impl. are achievable thru similarly-named methods | Assists developers in applying PF-style constraints but does not enforce them. Stylistic points of impl. are achievable thru selectors and channels vs. P/F |
| Matching Assumptions | Does not expect data/control will be piped by either a class in the same program, a different process/thread for a separate main(), standard in/out/err streams (for ie: program arguments). | Does not expect data/control will be piped by either a class in the same program, a different process/thread for a separate main(), standard in/out/err streams (for ie: program arguments). |
| Efficiency | Buffers can be used as wrappers for streams and asynchronous processing of streams might ensure i/o can occur separately from computation. | Buffers can be used as wrappers for streams and asynchronous processing of streams might ensure i/o can occur separately from computation. Mapping data(-sets) to native bytecode and operations to more efficient constructs within the OS ensure even more efficient I/O than the java.io threaded capabilities afford |