The Keck/3DFM microscope is an evolving system whose complex data acquisition processes require the careful coordination and automation of numerous instruments through computer software. A fundamental requirement of such a system is the composition of new processes to meet the growing demands of its users. Familiar programming practices however place conceptual limits on how easily and correctly one can express complex combinations of instrument actions. Thus realizing an effective programming model to enable this is a key concern in the further development of the microscope. This work presents a simple domain-specific programming language designed to alleviate the programming burden of the Keck Microscope’s data acquisition processes. The Keck’s instruments can be broadly classified into state and data devices and are represented as a form of finite state automata. The language addresses the problem of composition by allowing individual automata to be combined freely with one another by joining their state spaces and transition functions creating new automata that execute the desired sequences of instrument actions. Through sequencing and setting of a set of operators, a user can create an arbitrarily complex (and re-usable) sequence of instrument actions (so long as all states are known) while being spared the burdens of explicitly defining a flow of control and other elements irrelevant to the data acquisition process.

Introduction

• Carrying out an image acquisition process requires various devices to be programmed in the correct sequence of actions. The familiar approach is to only abstract the physical instruments as objects with the appropriate methods (such as “move”, “set” or “read”), then to define other constructs that directly manipulate them to implement image acquisition processes. This approach however does not yield an appropriate level of abstraction. The programmer must explicitly code the steps to set instrument states, retrieve image data, and define the flow of control for each and every process as well as test and debug the resultant logic to verify its correctness. Thus defining new processes using this low level of abstraction is an error prone undertaking requiring significant programming effort.

• This complexity however can be eliminated by treating each instrument as a simple, independent machine with a set of known states that will transition to a particular state (using a transition function) based on an input. Each state device has a sequence of states (which need not be every possible state for that device), and its transition function physically sets the device to the next state in the sequence, until the sequence is exhausted. A data device has no notion of state, and its transition function will always yield a datum (in this case, image data collected from a framergader).

• A data acquisition process therefore can be expressed as repeated iteration over a sequence of automatons where each automaton’s transition function is invoked once per iteration. Since every automaton knows how to perform all its respective actions, the programmer only needs to specify what actions are to be performed without concern for how they are to be performed. Thus the level of abstraction has been raised.

This work is supported in part by the Center for Subsurface Sensing and Imaging Systems, under the Engineering Research Centers Program of the National Science Foundation (Award Number EEC-9986821).

Implementation

• The central construct of the language is the automaton whose definition is derived from iterators and generators (Figure 1). The next() method is the transition function for state and data automatons as well as all other language constructs. For state devices, an automaton behaves as an iterator that physically sets the underlying device to each value being yielded and for data devices, an automaton acts as a generator providing an infinite stream of images.

• A framework is provided that allows the programmer to define instruments as combinable automatons and to define operators on them as well. For state automata, one subclasses StateDevice and defines the set() and get() methods which set the device to a particular state and retrieve the current device state respectively (Figure 2). For data automatons, one subclasses DataDevice and defines the next() method to yield an appropriate datum (Figure 3).

• Any arbitrary transformation can be applied to the elements being returned from an automaton. Users may thus define their own operators to suit their needs in addition to the standard set provided by the language.

Results

• Complex series of actions can be implemented using simple expressions (Figures 4, 5, 6).

• Complex acquisition processes can thus be decomposed into small, understandable, and re-usable series of expressions alleviating a significant source of error and programming effort.

Conclusions / Future Work

• Develop a full repertoire of automatons and operators to meet the needs of the Keck microscope.

• Develop a visual programming environment based on the concepts introduced here.

Opportunities for Technology Transfer

• The system and methodology introduced here can prove useful to other microscopy systems as well as other laboratories requiring a high degree of automated data acquisition.