Symphony requirements and architecture

Purpose

The purpose of this document is to document pre-construction planning for the Symphony data acquisition system. This document includes definition of the problem domain, requirements, acceptance testing plans and architectural designs for the application.

Problem description

Current physiology data acquisitions systems do not provide the combination of flexibility, integration with disparate systems, and high performance (i.e. pseudo real-time) required by the Rieke and Murphy labs:

1. The Rieke and Murphy labs perform elctrophysiological recordings (including real-time feedback dynamic clamp) using combined electrical, analog visual (e.g. LED) and rendered visual (e.g spatial stimuli presented on a computer monitor) stimuli. The existing solution (Igor/Acquirino) does not provide needed and expected flexibility in customization and does not easily integrate with rendered stimulus presentation. Existing solutions for dynamic clamp configuration recordings do not provide continuous recording capability.

2. In some experiments, recordings are combined with data from fluorescence imaging (confocal or two-photon). Acquirino does not provide for coherent storage of this external data and does not integrate with imaging acquisition software.

3. Existing systems do not provide all the required functionality while still being user-customizable for stimulus generation, experiment protocol and online analysis capability. Because experimental goals and techniques evolve over time, users must be able to customize stimulus generation and experimental protocol using high-level scripting languages (e.g. Matlab, Python, etc.). Currently users must modify the monolithic Acquirino code to customize stimulus generation or modify the Igor code to change experimental protocol.

4. Acquirino performs online analysis between trials, limiting the utility and breadth of online analysis capability. Like stimulus generation, online analysis requires modification of the monolithic Acquirino code base.

5. Acquirino does not provide unambiguous output suitable for import into a data management system. Parsing of a user-modifiable notes file is required to associate experiment metadata with recorded data.

Functional specifications

User Experience and High-level Program Behavior

Before an experiment, the user will configure the mapping from available external devices to A/D channels. Configuration may be via GUI or configuration file (yet to be determined). The configuration will be saved as a per-system application preference and restored the next time the application starts.

Commented [BW1]: Gabe suggests per-user configuration.

During an experiment, Symphony maintains a queue of epochs to present. This queue will be shown to the user via a GUI and the user will be able to edit this queue via keyboard or mouse interaction. The user may start or pause the queue at any time. If the user pauses the queue in the middle of an epoch, acquisition will stop and the incomplete epoch will be discarded. Some experimental protocols (e.g. Seal/Leak) may wish to present epochs that do not have a fixed duration. Stimuli for such epochs must be generated on-line and be streamable from the associated stimulus plugin. An epoch without a duration is called an "indefinite" Epoch. Indefinite epochs may not record experimental input (i.e. recorded data from an indefinite epoch will not be persisted), but online analysis plugins may request streaming data from indefinite epochs. Indefinite epochs will automatically be terminated when the epoch queue has ready epochs (i.e. as soon as the user enqueues epochs during a Seal/Leak epoch) if the queue was empty when the indefinite epoch began. The user may interrupt ("skip") an indefinite epoch at any time. Upon interruption, Symphony will begin the next epoch in the queue if one is available or stop acquisition if no epoch is available.

To add epochs to the queue, the user will select from a list of available protocols (as identified by the loaded protocol plugins). Settings for the selected protocol will be displayed for the user to edit. At the user's indication, Symphony will request the next block of epochs from the protocol plugin and will append those epochs to the queue.
The combination of protocol and its settings may be saved as a "Protocol Preset" that the user can recall for convenient access to commonly used protocol settings.

The UI will allow the user to specify a set of "sticky tags". These tags, if any are specified, will be applied to all epochs as they are sent to the symphony-core component for presentation.

The results (recorded results and optionally stimuli) will be shown to the user at the end of each epoch. Presentation types will be 2-D graphs, movies or images.

The user may select one or more online analysis plugins to run. Each epoch will be passed, after acquisition, to the online analysis in parallel with the persistence mechanism. As noted below, the purpose of the parallel persistence and online analysis pathways is to minimize the amount of time in which an exception or crash could prevent recorded data from being written to disk.

The online analysis plugins will have available an in-UI canvas for their use and/or may display separate UI for their results. Online analysis plugins will advertise the protocol types that they can analyze. The user may indicate which of these types they want to be analyzed by the plugin. After each epoch, the epoch will be passed, in parallel, to all online analysis plugins that advertise capability to handle epochs of the protocol of the finished epoch and for which the user has request analysis of that protocol type.

Inputs and Outputs

Symphony will produce experimental output via various D/A interfaces, video display and inter-process communication with external programs such as imaging systems. Experimental input will be from A/D interfaces as well as inter-process communication from external programs such as imaging systems. Video input will not be supported. See sections below for requirements of each subsystem. In this document, experimental input describes data that is recorded by the workstation. Conversely experimental output is the data presented to the biological preparation.

The fundamental unit of input/output will the "Epoch", a unit of time corresponding to a logical trial in the experiment. Because acquisition may be continuous, the Epoch marks trial boundaries though there may not be a corresponding boundary in the recorded data stream.

A/D hardware

Symphony will interface with the Instrutech ITC-18 data acquisition system as the primary A/D hardware and the canonical timeline clock (the ITC-18 internal clock will the the canonical time for the experimental timeline). Although the ITC system is the intended first target, Symphony will be architected in such a way as to provide for using other data acquisition systems in the future. Symphony will support all input/output channels supported by the A/D hardware. At minimum, this is 4 analog out, 8 analog in, 16 (in)/32 (out) synchronous digital in/out channels and 16 asynchronous digital output channels for a single ITC-18 interface.

D/A output will be provided by user stimulus/protocol plugins (see below) as typed numeric arrays (e.g. IEnumerable<T>) with associated physical units (e.g. volts) and rendered visual output as calibrated bitmap data and associated physical units. Experimental input (A/D input) will be converted form ITC driver to typed numeric arrays (e.g. IEnumerable<T>) with associated physical units.

In case of exception in hardware acquisition Symphony will stop data acquisition on all channels. Symphony will present a visual indicator to the user describing the source of the exception. Symphony will return the IO subsystem to the ready state (i.e. for re-start with the next available epoch). Epochs during which an exception occurs will be discarded (i.e. not written to disk).

At acquisition stop (by user command, end of epoch presentation or exception), all A/D outputs and video outputs will be set to the last "background" value in the last presented

Epoch. In the case of an exception, the background values specified by the epoch whose presentation caused the exception will be used.

Values outside of the presentable range of the A/D hardware will generate an exception in the Symphony output pipeline.

Video output

Symphony will provide time-locked video output via the workstation's standard video hardware. Rendering will be via OpenGL or DirectX commands and frame presentation will be time-locked with the experimental timeline (see below). Symphony will support a maximum of one experimental output display.

Commented [BW2]: StimGL may already support this

Interprocess communication

Symphony will communicate with external programs such as imaging or presentation systems via interprocess communication (e.g. COM). Interop capability will depend on the API supported by those external processes and has not yet been determined.

In the case of an exception in inter-process communication, Symphony will present a visual alert indicating the cause of the exception. Data presentation and acquisition will continue as normal without interruption following an exception if the result of the communication is not critical for acquisition. In the case that interprocess communication is required to continue acquisisiton, Symphony will stop acquisition, persisting all recorded data to disk and discarding incomplete epochs. Acquisition can then be restarted from the beginning of the next epoch. Communication may be re-tried following an exception.

(Optional) Virtual simulator

Symphony will provide a "simulation" mode whereby experimental output is directed to a simulator component and experimental input is read from that component. Simulation mode is intended to allow computational modeling experiments to be integrated into the standard lab workflow. For example, a simulation testing a computational model of cell response to a stimulus could be run using the same protocol and stimulus generation code as was used to record actual cellular responses. The output of the simulation would be in the same format as that from the experiment, further allowing comparison of experimental and modeling results.
Simulators will be loaded as plugin components, conforming to a prescribed API. D/A output will be provided as typed numeric arrays (e.g. IEnumerable<T>) with associated physical units (e.g. volts) and rendered visual output as calibrated bitmap data and associated physical units. Experimental input (A/D input) will be received form the simulator as typed numeric arrays (e.g. IEnumerable<T>) with associated physical units.

External device description

Symphony will support description of external devices via loadable plugin (e.g..Net Managed Extensibility Framework). An external device provides information on the physical units of the device (e.g. mV for experimental input from a current clamp amplifier). Epoch descriptions (see EXPERIMENTAL PROTOCOL below) will specify channels for experimental input/output via these device descriptions. Device may be associated with particular hardware data channels via user-configurable UI or configuration file interface.

Symphony will initially ship with support for the external devices currently in the Murphy and Rieke labs. These include the MultiClamp amplifier, LED controller, and the Warner Temperature controller. The AxoPatch amplifier will not be supported.

Continuous Acquisition

Symphony will be able to present stimuli and record results without gap between epochs (i.e. without D/A underrun between epochs). In the case that an epoch is not marked continuous, acquisition will stop and restart before that epoch. In the case that stimulus generation (or other necessary tasks) are unable to provide a fully ready epoch before the previous epoch completes, acquisition will stop and restart once the next epoch is complete. An epoch is complete when its stimuli are fully rendered. A stimulus is considered "rendered" when the stimulus plugin responsible for its generation indicates that the stimulus is ready and can be streamed (as IEnumerable<T>) without iterruption.

Real-time feedback

Symphony will provide for real-time feedback between experimental output and input. For example, the experimenter may wish to calculate output in real-time given real-time inputs in a dynamic clamp experiment. **Real-time feedback will be implemented by external programmable hardware.**

Real-time calculation will be specified by a "kernel" that can be executed on the CPU or on Arduino-style external hardware. Symphony will support download of the real-time kernel to a specialized hardware device via USB, according to the Arduino API (or similar). Input to the kernel hardware device will be via the standard (i.e. Instrutech) D/A output.

Symphony will optionally provide for execution of a "kernel" on the workstation CPU (as part of the standard data acquisition loop) but will not guarantee pseudo-realtime performance of this pipline.

Experimental timeline

Symphony will record and present experimental input/output on an experimental timeline, accurate to 1 micro-second. All input-output modalities will share this common timeline. Persistent data will be recorded and time-stamped relative to this experimental timeline.

Stimulus generation

Symphony will generate experimental output (i.e. stimuli) via loadable plugins (e.g. MEF) that provide a parameterized generation function. Analog and digital output data will be generated by plugins as typed numeric data (e.g. IEnumerable<T>) with associated physical units. Symphony will convert these physical units to the appropriate D/A voltages in the output pipeline.
A/D will be "rendered" before their associated epoch is presented. Rendering will consist of generation of a byte array (in memory) via a stimulus-generation plugin.

Rendered visual stimuli will be rendered as OpenGL at presentation time. Rendered visual stimuli will be given a chance to pre-compute necessary values before presentation. This pre-rendering computation will happen along side rendering of A/D stimuli.

Stimulus values outside of the presentable range will generate an exception at presentation time. There will be no error checking of out-of-range values at rendering time.

Symphony will be able to continue acquisition without interruption in the case of an exception within the stimulus generation plugin code. Exceptions will be logged and occurrence of the exception presented to the user. When an exception occurs, the stimulus not be rendered and thus the associated epoch will not be ready to present. When this epoch becomes the first epoch in the presentation queue, acquisition will stop and the user will be notified via UI of the reason acquisition stopped (i.e. because the stimulus could not be rendered). The user may delete the offending epoch and then restart the acquisition queue.

Experimental Protocol

Symphony will delegate experimental protocol description to loadable plugins (e.g. MEF). This protocol will consist of definition of each epoch to be presented, including which channels to use for input, output, stimulus generation parameters, hardware modes (e.g. voltage-clamp, current-clamp, etc.), real-time kernels for each channel and necessary information about epoch continuation (e.g. continuous etc.). In addition, the Epoch must specify a value for "background" on each output channel/modality to be used in case no stimulus is defined for that channel but the channel is marked as active for that epoch.

The protocol plugin will be asked for epochs in "blocks." For example, a flash family protocol will provide epoch descriptions of the entire flash family with the request for a block of epochs. A protocol may apply any keyword tags to epochs before returning the epoch block. Each block will be provided as an IEnumerable<Epoch>. Thus the protocol must provide IEnumerable<IEnumerable<Epoch>>.

An epoch must indicate its total duration. This duration may be indefinite, in which case the epoch will be terminated at user request or when the epoch queue has available epochs but was empty when the indefinite epoch began.

An epoch may optionally specify that Symphony should return to the indeterminate epoch that was previously executing, if one exists. Thus if Seal/Leak is running, a flash family may specify that a Seal/Leak epoch be run after the flash family block completes.

Changes in active channels will produce a stop/restart of acquisition between epochs.

Protocol settings for a protocol will be persisted as per-user application preferences and applied at application startup to each loaded protocol.

In case of exception during epoch generation, the epoch will be discarded. Symphony will present a visual alert indicating the cause of the exception. Data presentation and acquisition will continue as normal without interruption following an exception in Epoch generation.

Online analysis
Symphony will optionally perform online analysis of recorded Epochs and present the results of this analysis to the user as it is completed. Online analysis must not interrupt recording or stimulus generation activities. Ideally, online analysis will not interrupt durable persistence of data to disk. For this reason, online analysis may not modify the data that will be persisted. Online analysis will be specified and performed via loadable plugins (e.g. Managed Extensibility Framework). Online analysis plugins may present their UI in separate windows or in a designated view container within the main Symphony UI (yet to be determined).

Symphony will provide a single Epoch description instance to the plugin. Plugins are responsible for retaining any required accumulation or state information, though they may retrieve previous epochs from the persistent store.

In case of exception in online analysis, the online analysis system's state will be returned to the state previous to analysis of the epoch during which an exception occurred. The user will be presented with a visual indicator describing the cause of the exception. Optionally, the user may suspend execution of the analysis plugin that caused the exception.

Online analysis plugins must be able to request a collection of epochs given a time range. This collection may be restricted by protocol type(s) if desired. These epochs must be available even if they have already been persisted to disk. Any changes made to these epochs, like during primary online analysis, will not be persisted.

Persisted data
Symphony will persist a complete and unambiguous record of the experimental timeline. Experimental input will be persisted as numeric arrays with associated physical units, in natural physical units (e.g. mV/pA for current-clamp and voltage-clamp recordings). See DATA MODEL below for the object-level description of persisted data.

No information from online analysis will be persisted.

Non-functional and performance specifications

Real-time (dynamic clamp acquisition)
Dynamic clamp acquisition requires that analog output be specified for each sampled input with minimum sampling rate of 40 kHz. The target sampling rate is 100 kHz for dynamic clamp on one analog input/one analog output channel.

Task Priority
Symphony will operate such that data presentation and acquisition receive sufficient resources to continue uninterrupted, at the expense of stimulus generation, UI presentation, and online analysis. Data persistence will receive next highest priority. All other tasks will have priority lower than presentation/acquisition and data persistence, possible at the expense of UI responsiveness or the ability to acquire data continuously.

Supported system hardware, and Operating System

Symphony will support Windows 7 with .Net 4.0 as the minimum OS/.Net version. Minimum system performance characteristics are not yet defined. Because the Instrutech driver is currently 32-bit only, 32-bit Windows is the initial target.

Network independence

Symphony must operate independent of network resources and must continue operation (including recording data and presenting stimuli) in the case of network interruption even if network resources were in use at the time of network interruption.

Disk usage

On-disk space requirements for persisted data will affect backup costs. Original persistent data will be archived indefinitely. Because persistent data will be imported into a data management system before analysis, on-disk space requirements will be prioritized above read performance for the persistent storage format.

Security

Plugins will run will full application privileges (i.e. without sandboxing). Any incoming network connections will be protected from unauthorized access by password and secured with public/private key cryptography (e.g. SSL). Symphony data is considered non-secret and no extraordinary measures will be made to protect application settings data or recorded results.

Fault Tolerance

In general, the fault handling strategy within Symphony will follow the layered architecture of the program. The goal is to prevent presentation of faulty stimuli or persistence of faulty recordings at all costs, including interruption of the experiment. Thus faults in the acquisition core will terminate acquisition and discard the data associated with the fault (e.g. data recorded after an exception). Faults in stimulus rendering or epoch generation will cause the associated epochs to be unpresentable. Faults in online analysis or UI presentation will be considered non-critical and will be ignored to the extent possible.
All faults will be automatically logged and optionally reported to Physion's issue tracker for support follow-up.
The persisted data store will be fault tolerant to software exceptions in the Symphony process. All non-faulty data recorded will be persisted to disk in the case of a fault anywhere in the Symphony process. The persisted data store will not be fault tolerant to errors in the operating system or workstation hardware.

Minimally useful deliverable

The minimally useful deliverable is a system that faithfully replicates the current Acquirino functionality. This system may be realized by replacing the current Acquirino core (written in C as Igor XOP extensions) with the Symphony Core via an XOP bridge. This may require a C or C++/CLI wrapper on Symphony Core and a C XOP on the Igor side.

Architecture

Program Organization

Symphony is organized according to an MVC architecture with a central ExperimentController class that coordinates the experiment, a layer of UI components, and a collection of data model classes (see below). The ExperimentController class performs experimental I/O via symphony-core. Symphony-core handles interaction with the A/D hardware, video presentation hardware, and other external devices.

Commented [BW3]: Technical comment

Within symphony-core, a pipline (producer/consumer) model will be used to transform outputs to the eventual device format and from the device input to eventual physical units. This pipeline will allow for inclusion of post-processors (e.g. spike threshold detection etc.) in the input pipeline in future iterations.

A possible realization of this input/output pipeline is via .Net's IEnumerable and IObservable and the Reactive Extensions framework or the F# pipeline model. The output pipeline will be "pull" from the Controller to IODevice. The input pipelines will be push from the IODevice to the Controller.

ExperimentController

ExperimentController maintains a queue of available epochs and implements Symphony.Core.IEpochSource. The ExperimentController has a reference to a Symphony.Core.Controller instance and serves as the epoch source for that instance. Delegates plugin handling to a plugin manager and/or MEF. Delegates preferences and stored protocol settings to a preferences manager.

DataDescription

DataDescription encapsulates an IEnumerable<T> of data samples, physical units of the data (dimension and exponent scale), a timestamp (on the canonical timeline), a dictionary (IDictionary<string,object>) of properties of the originating endpoint and a list (IEnumerable< {name, IDictionary<string,object>}>) of names and properties of intervening IONodes.

Video output

Video output for the Murphy lab will be via the StimGL system. Video output system for the Rieke lab is yet-to-be determined. Support will be for a single output system. Resolution (e.g. multiple-monitor) or frame-rate scaling can be achieved via hardware systems such as the triple-head system and techniques used by the StimGL system to multiplex frames within a single DVI frame packet.

UI and Plugins

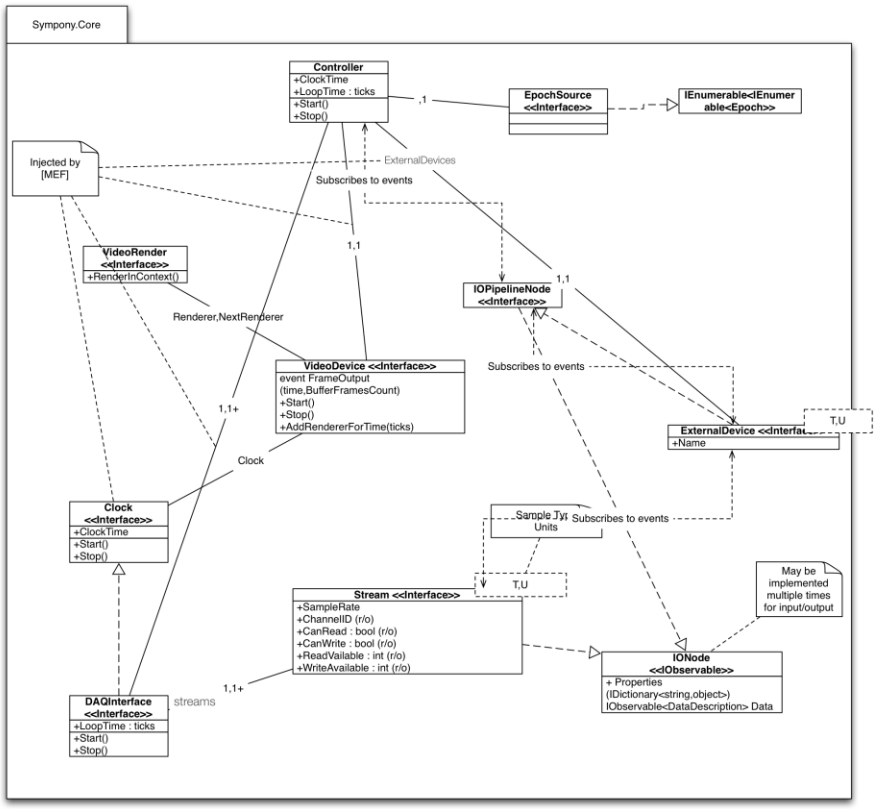
The Symphony UI will be developed in .Net WPF.

Plugins will be provided as MEF components, optionally using the Dynamic Language Runtime (IronPython or IronRuby) or F#/C#. Optionally, stimulus generation and protocol description may be provided as Matlab functions/classes via the Matlab COM server.

~~sympony-core~~ Symphony.Core

Symphony ~~core~~ Core abstracts the presentation and recording of Epoch data. Experimental output is streamed from its source by symphony-core. Experimental input is streamed into an Epoch instance and associated Response instances. The fully complete Epoch is returned to the ExperimentController upon completion. At high-level, symphony-core thus represents a set of parallel data pipelines. Each external device identifies one of these pipelines. A pipeline may pass data for output (i.e. to the experimental preparation) or input (i.e. from the experimental preparation). The controller will push data into output pipelines in response to events signaling data output from the associated external device such that approximately 200ms buffer is maintained in the output pipeline. The DAQInterface will push data into input pipelines when data is received from its analog or digital channels.

The proposed class diagram sketch for symphony-core is shown in below:



Symphony Core will be implemented in .Net. All calculation sin the input and output pipelines will verify dimensional correctness of the results. F# provides this capability but is not the required implementation language.

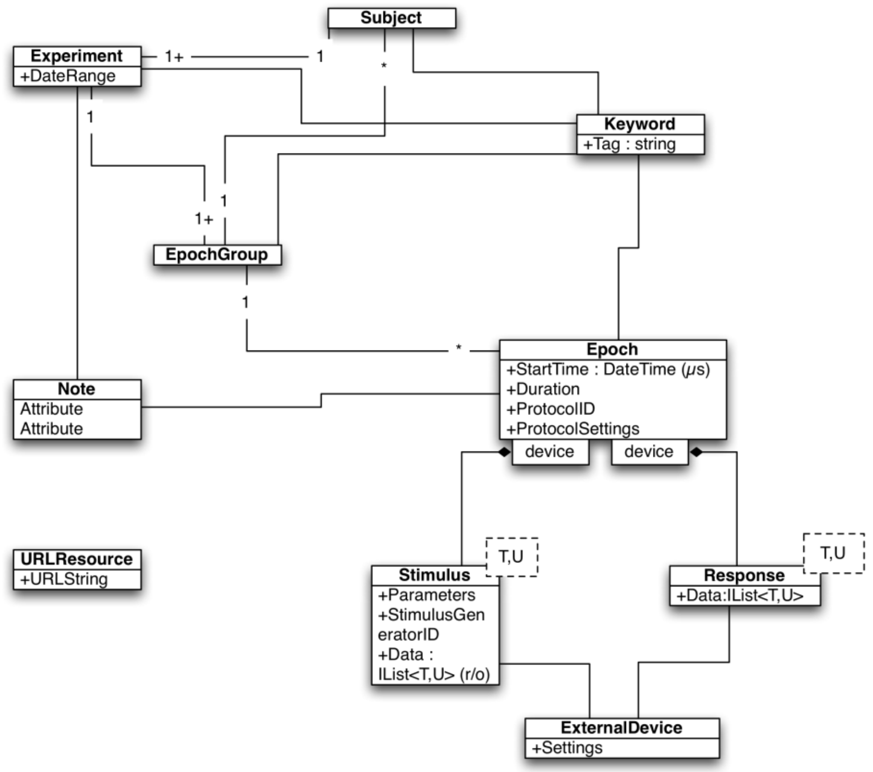
ITC-interaction will be by the "Universal" driver. Performance of the universal driver, in particular maximal iteration rate, will be verified by a testing spike in the AcquirinoSandbox project as part of Iteration 0. In case the universal driver is not performant, the ITC-18 driver will be used in symphony-core's ITC interaction. In this case, the initial release of Symphony will support the ITC-18 *only*. Use of the ITC driver will naturally require a P/Invoke or C++/CLI bridge.

Because the ITC driver is synchronous (polling), it the ITC implementation for of IDAQInterface will necessarily have to run in a separate thread. With this exception, the reactive paradigm (for input) and pull (i.e. IEnumerable; for output) appears appropriate for the rest of Symphony-Core. Possible implementations are raw .Net events, though Rx or F#'s pipeline model appear higher-level and more appropriate. In the case of events or Rx, an appropriate SynchronizationContext will be required to provide events on the appropriate (i.e. Controller and pipeline) thread. For multiple pipelines (input/output for each stream endpoint), events can be processed in parallel as there is no interaction between the pipelines associated with different stream endpoints. In the case of parallel processing, each node must appropriately timestamp its activities so that data can be reassembled in appropriate order (and assigned to the appropriate Epoch) at the Controller.

The F# data pipeline and or the .Net IEnumerable and IObservable (Reactive Extensions framework (Rx)) provide natural foundations for the input/output pipelines of symphony-core.

Data model

The Symphony data model mirrors the Core Data model from the Symphony prototype in the rieke-server SVN. The model is shown below:



Epochs will also contain references to previous and next epochs on the experimental timeline.

Persistence mechanism and format are yet to be determined. HDF5 and SQLite backends are under consideration. The advantage of HDF5 is that it is readily readable by many client libraries. It may be difficult to match the Symphony data model to the HDF5 data model, however. SQLite is an obvious choice as was the backend for the original Symphony prototype (via Apple's Core Data). SQLite is similarly readable from common analysis targets (e.g. Python, Matlab, Mathematica) but its use is significantly more complicated than HDF5 for non-programmers. An ORM layer (e.g. SQLAlchemy, Mono's SQLite provider for Linq-To-SQL, or a SQLite provider for Entity Framework) layers on top of SQLite might bridge this gap. For import into a data management system, either target is acceptable.

On-disk space usage will be dominated by raw data. Because Symphony will support raw data in arbitrary formats and with arbitrary pre-processing transformations (e.g. detecting threshold crossings for extra-cellular recordings), on-disk format will be in physical units. Following Boost.Units convetion, physical units will be defined as a base unit (dimension

and system) and a scale (e.g. log(10^x) where 10^x is the multiplicative scale factor between base and derived unit. SI will be the on-disk standard for unit system. Support for non-SI systems could be added in the future, but initially, Symphony will require SI units from plugins (e.g. stimulus generation).
Storing data in, e.g. DAQ counts would save 16-bits per sample, but would require all API users to be aware of the DAQ conversion and would not generalize to pre-processed data or data that was not acquired from an A/D converter.

For data acquired from an A/D converter, a floating point format of sufficient precision to represent the entire range of DAQ counts will be used. For a 16-bit DAQ count, this corresponds to a 32-bit IEEE float, with associated physical units. To allow recovery of the original DAQ count from the on-disk format, conversion information from DAQ counts to physical units will be recorded with channel information for each A/D channel for each epoch.

Feasibility and Risk

The major risk associated with the Symphony project are the requirements for dynamic clamp performance and common timeline support.

Dynamic clamp places significant latency and performance requirements on several components of Symphony. These constraints on performance and low-level control of the acquisition loop are in conflict with the requirements of continuous acquisition and easy user customizability via high-level (i.e. potentially low-performance) languages. There is a significant risk that Symphony will not be able to perform dynamic clamp using software-only solutions. The addition of an Arduino-style hardware device to perform the dynamic clamp feedback loop significantly mitigates this risk.

Testing of the Win32 port of CJ Bell's GCProfile project has allowed testing of dynamic clamp performance of the Windows OS/Magma/ITC-18 system. Testing verifies that 40kHz sampling with 0 buffer under-runs is achievable. Thus, a CPU-only fallback in the case of failure of the hardware system is viable.

Common timeline support for stimulus presentation and recording is critical for experimental protocols that use multiple hardware interfaces that do not share a common A/D interface. Spatial stimuli is the prime example of this challenge as the targeted A/D interface and the host CPU use separate clocks. The Symphony architecture will use the A/D interface clock as the canonical timeline clock. The challenge then becomes synchronizing spatial stimulus presentation to this clock. Hardware triggering of display refresh via a signal from the ITC interface is the most secure method of achieving this synchronization. If this hardware synchronization is not possible, there is risk that Symphony will not be able to maintain a consistent common timeline for experimental output.

All other components in Symphony are comparatively standard elements and do not pose significant risk.

Minimizing risk
The construction plan for Symphony will minimize the risk associated with both acquisition loop performance and common timeline support. Conveniently, the major risks in the project are associated with the symphony-core component. Thus a bottom-up integration plan is also a risk-oriented integration plan. In bottom-up integration, the lower-level components are built first and tested. Then higher level components (UI etc.) are build and integrated with the lower-level components. Because risk grows as the time between implementation and testing, being able to detect errors in the architecture early will that affect the risk-prone areas of the project.

Scalability
Symphony will operate as a multi-threaded application. Data acquisition will be performed on a dedicated, real-time priority thread (and necessary subsidiary threads). Stimulus generation, online analysis, data persistence, inter-process communication (e.g. with imaging systems) and UI presentation will be provided on individual threads.

A multi-threaded architecture allows scalability of the system performance with additional processing cores and allows fine-grained priority assignment for each task in the system. Additional cores may be especially helpful in adding worker threads for stimulus pre-generation and online analysis.

Online analysis may be further scaled by using distributed computing resources. In this extension, online analysis tasks are farmed to local or remote computational cluster resources for analysis and the results collected and displayed by the online analysis thread in Symphony.

Testability
The most difficult component to test will be rendered stimulus presentation. All other components may be tested either via simulation (e.g. ITC loopback device), standard unit testing strategies or on a physical loopback (i.e. ITC18) device.

Rendered stimulus presentation will be tested by manual inspection of generated movies.

We will strive for 100% code coverage with unit tests excluding spatial stimulus rendering code. Implementations of IODevice and Stream in Symphony.Core for the ITC device will be unit tested using the loopback interface of the ITC driver. Functional testing of Symphony.Core will be performed via automated buildbot testing on the USB/PCI hardware.

Integration testing will be performed to the level of ExperimentController. UI testing will be by manual testing.

~~Initial Roadmap and Acceptance Testing~~