| **№** | **Краткое описание** | **Речь** |
| --- | --- | --- |
| 1 | Тема, ФИО | Уважаемые члены Государственной Аттестационной комиссии, хочу продолжить вашему вниманию вторую часть дипломного проекта, первую часть которого только что представил мой коллега. |
| 2 | Консоль | В первой части было представлено ядро системы. Оно по идеологическим причинам работает в консольном режиме. Этот режим очень удобен для запуска и функционирования, но при этом он не очень удобен для восприятия человеком. Ведь в человеческом мозгу нет такой единицы хранения информации как слово. Мозг все запоминает образами, в большинстве своем визуальными. |
| 3 | Конфигурационные файлы | Для управления консольными приложениями часто используют конфигурационные файлы. Но у этого способа есть ряд недостатков. Конечно, некоторые из этих недостатков можно решить дополнительными средствами или определенными условиями. (Разве консоль управления не является доп. средством?) Например, решить проблему конфигурирования нескольких приложений из одного места можно, сделав доступным им всем один файл конфигурации с удаленного сервера. Это возможно… ВОЗМОЖНО решит одну проблему, но возникнет ряд других (каких-то) неудобств. |
| 4 | Структура проекта -  страшная схема | Для решения этих проблем (и не только)(а каких еще?) и необходима вторая часть дипломной проекта. Само по себе ядро может проделать большую работу, но для того, чтобы получить выгоду от этой работы, необходимо еще и эффективно использовать результаты этой работы. Поэтому основная задача второй части работы заключается в организации взаимодействия ядра с окружающим миром. Первым пунктом является **Панель управления** – инструмент управления работой исполняемой среды и визуализации информации процесса и результатов этой работы. Вторым - **Прикладной интерфейс программирования** позволяет разрабатывать модули мониторинга на основе унифицированного каркаса исходного кода модуля. |
| 5 | Модель приложения MVC | В основу модели приложения Панели управления был положен шаблон проектирования, называемый MVC или Model-View-Controller. Основная идея этого шаблона – разделение приложения на три (у вас только три?) основных компонента с «слабой» связанностью между ними. Компонент Модель реализован классом **домен**, который содержит информацию о ядре и логику получения, обработки и предоставления этой информации. К компоненту Представление или интерфейс можно отнести класс **Главное окно**, которое является пользовательским интерфейсом в классическом понимании этого выражения. Управляющий компонентом Поведение является класс **координатор**, который согласовывает информацию из домена и ее отображение в пользовательском интерфейсе. |
| 6 | Диаграмма последовательностей | Сейчас Вы видите общую схему функционирования Панели управления и ее взаимодействия с ядром. Ядро через свой драйвер Исследователь оповещает панель о своей активности через определенные промежутки времени. Так же с помощью этого вызова передается минимальная информация о ядре, так называемый контекст ядра. На основе которого делается вывод об изменение ядра. Если изменение произошло, то домен обновляет локальную информацию и оповещает об этом интерфейсу, который обновит отображение. Если же пользователь совершит какое-либо действие, то через систему отлавливания событий об этом сразу станет известно координатору, который организует соответствующие изменения в домене. После изменений в ядре информация в домене обновляется, о чем оповещается интерфейс. |
| 7 | Архитектура приложения  диаграмма классов | Архитектура приложения включает в себя множество классов, которые условно можно отнести к трем основным компонентам. При рассмотрении заметно, что каждый компонент MVC реализован не одним классом, а несколькими. Но в каждой группе выделяется один основной представитель, который и олицетворяет реализованный компонент |
| 8 | Хранение информации  Домен | Минимальная информация о ядре системы и отдельных узлах, которая визуализируется пользователю, хранится в динамических массивах, списках, которые в свою очередь содержатся в контейнере Domain. Хранится локально в ОЗУ на узле, где запущена панель управления. При этом она не сохраняется в ПЗУ, а в случае перезапуска панели запрашивается у ядра повторно. Домен содержит адаптер Исследователь, который предоставляет ядру интерфейс, через который контекст доставляется панели. Поскольку в домене храниться только минимальная информация вся дополнительная получается через интрейфейсы. Информация кэшируется и если она требуется, то берется из этого кэша, а не запрашивается у ядра каждый раз за исключением ситуаций, когда необходимо принудительно обновить информацию. Такой подход позволяет снизить уровень трафика сетевого взаимодействия и повысить отзывчивость панели управления. |
| 9 | Взаимодействие компонентов –  Координатор | Основной задачей координатора является установление соответствия между действиями пользователя и изменением информации в домене. Таким образом получается, что снаружи взаимодействие Домена и Интерфейса выглядит очень прозрачно. |
| 10 | Визуализация –  Главное окно | В качестве основы для графического интерфейса была взята концепция многодокументного интерфейса (MDI). Такая организация интерфейса очень удобна, если необходимо работать с большим количеством однотипных окн. Подразумевается, что в панели управления будет открыто много окн, поэтому такая архитектура была выбрана, как наиболее подходящая.  Таким образом есть главное окно и множество внутренний окн. При выполнении пользователем каких либо действий с интерфейсом, произойдут действия которые соответственно определены в координаторе. |
| 11 | Использование API  кусок кода | Одной из ключевых особенностей панели управления является прикрепление модулей к узлам. Для удобства и универсальности был разработан соответствующий прикладной интерфейс программирования или API. **Прикладной интерфейс программирования** позволяет разрабатывать модули мониторинга на основе унифицированного каркаса исходного кода модуля. Этот интерфейс задает правила исполнения, передачи параметров и возврата результата модуля. В текущей реализации интерфейс программирования модулей представляется каркасом с одним публичным методом – «invoke(..)». Параметры модуля мониторинга могу передаваться как обычная коллекция или список языка Python. Помимо интерфейсного метода invoke в модуле могут быть и внутренние методы. |
| 12 | Развертывание модуля  3 скриншота | Сам процесс развертывания модуля довольно просто. Все что необходимо, это, использую встроенный в панель управления текстовый редактор, написать модуль использую АПИ. Хочу заметить, что модули не обязательно писать самому. Их можно, например, скачать в интернете. Когда есть модуль, достаточно выбрать список узлов, на которые необходимо развернуть модуль, и нажать кнопку Развернуть (Deploy). Теперь модули развернуты и можно получать результат их работы. |
| 13 | Итоги | За время выполнения дипломной работы была выполнена большая работа. Программированию алгоритмов и моделей предшествовала исследовательская и аналитическая работа. |
| 14 | Спасибо! Вопросы? |  |

Я сделал!

Профессионально!

Много!