МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

СОГЛАСОВАНО
Генеральный директор
ЗАО «АйТи»

_____ Бакиев О.Р.
"29" декабря 2011 г.

УТВЕРЖДАЮ
Ректор НИУ ИТМО

_____ Васильев В.Н.
"29" декабря 2011 г.

МНОГОПРОФИЛЬНАЯ ИНСТРУМЕНТАЛЬНО-ТЕХНОЛОГИЧЕСКАЯ ПЛАТФОРМА СОЗДАНИЯ И УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ СРЕДОЙ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ CLAVIRE

ПРОГРАММНЫЙ КОМПОНЕНТ МОНИТОРИНГА CLAVIRE/MONITORING

ОПИСАНИЕ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.80066-06 13 24-ЛУ

Представители
Организации-разработчика

Руководитель разработки,
профессор НИУ ИТМО

_____ Бухановский А.В.
"29" декабря 2011 г.

Ответственный исполнитель,
с.н.с. НИУ ИТМО

_____ Луценко А.Е.
"29" декабря 2011 г.

Нормоконтролер
ведущий инженер НИУ ИТМО

_____ Позднякова Л.Г.
"29" декабря 2011 г.

2011

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДЕН
RU.СНАБ.80066-06 13 24-ЛУ

МНОГОПРОФИЛЬНАЯ ИНСТРУМЕНТАЛЬНО-ТЕХНОЛОГИЧЕСКАЯ ПЛАТФОРМА СОЗДАНИЯ И УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ СРЕДОЙ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ CLAVIRE

ПРОГРАММНЫЙ КОМПОНЕНТ МОНИТОРИНГА CLAVIRE/MONITORING

ОПИСАНИЕ ПРОГРАММЫ

RU.СНАБ.80066-06 13 24

ЛИСТОВ 20

2011

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

АННОТАЦИЯ

Документ содержит описание программного компонента мониторинга CLAVIRE/Monitoring RU.СНАБ.80066-06 01 24, в рамках процесса динамического управления распределенными ресурсами отвечающего за сбор данных о работе многопрофильной инструментально-технологической платформы (МИТП) CLAVIRE и предоставляющего ее компонентам информацию о состоянии вычислительной среды, в которой МИТП функционирует. Программный компонент мониторинга разработан в ходе выполнения «Создание распределенной вычислительной среды на базе облачной архитектуры для построения и эксплуатации высокопроизводительных композитных приложений» (Договор № 21057 от 15 июля 2010 г., шифр 2010-218-01-209) в рамках реализации постановления Правительства РФ № 218 «О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства».

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Компонент мониторинга (КМ) CLAVIRE/Monitoring RU.СНАБ.80066-06 01 24 призван в рамках процесса динамического управления распределенными ресурсами осуществлять сбор, хранение, обработку и предоставление информации о платформе в целом и ее компонентах. Помимо основной функции сбора и представления информации КМ может обеспечивать сервисные механизмы для разработки и контроля эксплуатации распределенной системы, поддержки пользователя при выявлении критических ситуаций, а также отладки системы. Данный компонент не зависит от других компонентов комплекса, а заложенная в его архитектуру расширяемость, за счет возможности разработки и динамического подключения дополнительных сборщиков, позволяет использовать его в качестве универсального средства мониторинга распределенных систем.

Данный компонент разработан на языке C# и требует для своего функционирования наличия следующего системного программного обеспечения: ОС семейства Windows NT (версии старше Windows 2000), .NET 4.0, Internet Information Services (с версией старше 6.0). Хранилища данныхреализованы на базе СУБД MongoDB, поэтому для функционирования данного компонента необходимо наличие развернутой базы данных MongoDB (с версией не менее 1.6.5).

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

КМ предназначен для выполнения следующих функций:

1) сбор информации о состоянии компонентов платформы CLAVIRE и используемых ей вычислительных ресурсов;
2) конвертация собранной информации в унифицированное представление;
3) первичный анализ поступающих в компонент данных;
4) хранение актуальной информации о состоянии комплекса, а также истории изменений состояния;
5) хранение информации о выполняющихся композитных приложениях;
6) предоставление интерфейса доступа другим компонентам и модулям комплекса к хранимой актуальной информации и истории изменений.

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1. Программная архитектура и принцип функционирования

Процесс мониторинга подразумевает под собой получение данных из различных удаленных источников посредством сетевого взаимодействия по определенным протоколам. По способу получения данных процесс сбора может быть разделен на активный (периодический опрос) и пассивный (прослушивание, подписка). Постоянный сбор и актуализация информации позволяют наглядно демонстрировать пользователю ход исполнения приложений, администратору отслеживать текущее состояние системы, компоненту планирования исполнения WF эффективно составлять расписание запуска задач. Хранение истории всех изменений в системе дает исчерпывающую информацию для анализа эффективности работы комплекса, отладки его конфигурации. На основании этой информации можно производить оценку загрузки тех или иных ресурсов, активности пользователей.

Компонент мониторинга представляет собой программное средство для сбора, обработки и представления информации о состоянии платформы. В состав КМ входят модуль сбора данных (*CollectionDaemon*), вспомогательные хранилища данных и интерфейс доступа к компоненту мониторинга (рис. 3.1).

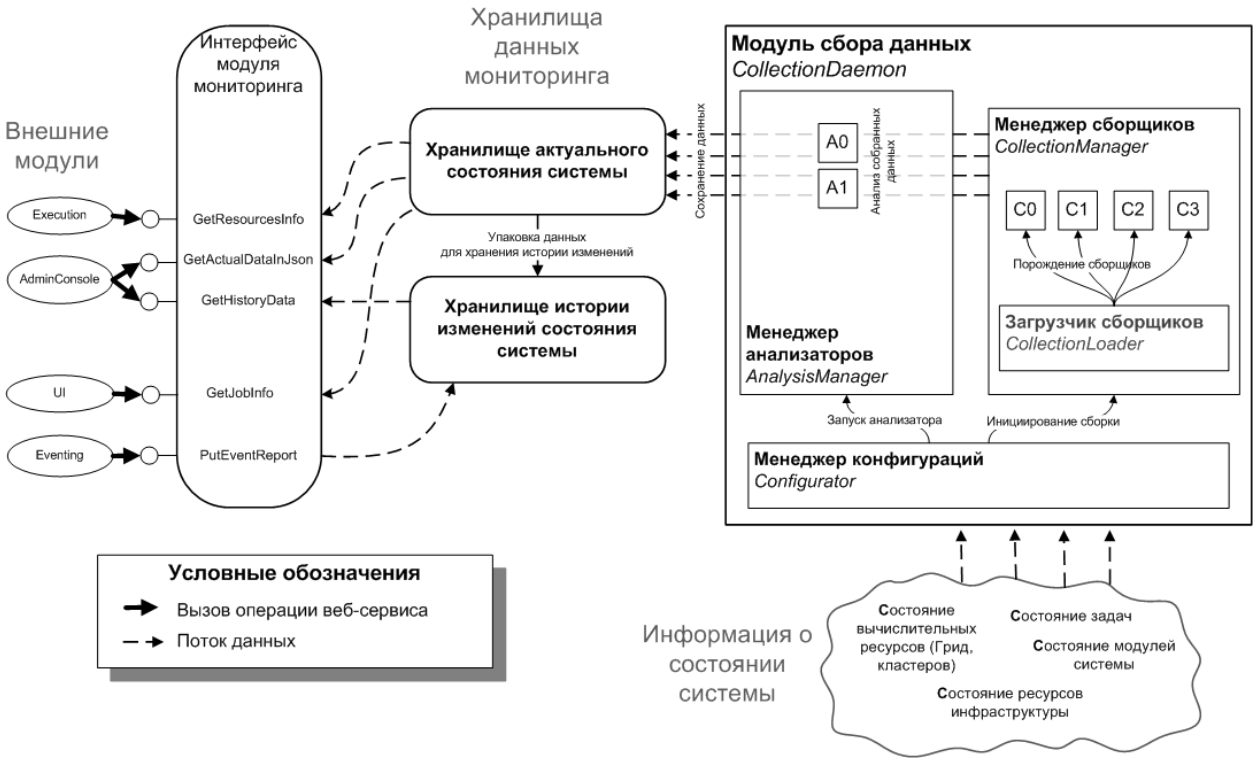

Рисунок 3.1 – Архитектура компонента мониторинга

Модуль сбора данных является основной частью КМ и отвечает за сбор, обработку и предварительный анализ информации. Источниками сбора данных для модуля являются: компоненты системы, вычислительные ресурсы, а также ресурсы, относящиеся к инфраструктуре платформы. Результатом работы модуля является непрерывное наполнение хранилища данными об актуальном состоянии системы, приведенными к унифицированному виду и помеченными информацией о времени их получения. Основной рабочей единицей модуля сбора данных является сборщик (или коллектор). Сборщик – это программа, которая позволяет непрерывно собирать информацию с определенного источника. Как уже упоминалось ранее, процесс сборки может быть как активным, так и пассивным. В качестве активного можно привести сборщик, периодически проверяющий доступность указанных компьютеров за счет посылки эхо-запроса ICMP (*PingCollector*). Примером пассивного может служить коллектор, который собирает информацию только по приходу оповещения об определенном событии от компонента событийного взаимодействия RU.СНАБ.80066-06 01 23.

Каждый активный коллектор (наследник *CollectorBase*) выполняется в собственном потоке. Для управления активными коллекторами используется менеджер сборщиков (*CollectorManager*). Для инициализации всей системы сборки используется менеджер конфигурации (*Configurator*), он выполняет начальную настройку модуля и дает команды на загрузку сборщиков для указанных целей сборки. Загрузку соответствующих сборщиков выполняет загрузчик сборщиков (*CollectorLoader*). Запуск коллектора предполагает его корректную настройку за счет заполнения данными и указания соответствующего ему контекста. Контекст – это набор конфигурационной информации, необходимой для работы коллектора. Каждый коллектор помимо контекста характеризуется набором поддерживаемых целей сборки. Цель сборки – это строка вида ”<тип информации>:<метод сбора>”.

В табл. 3.1 перечислены доступные типы сборщиков и поддерживаемые ими цели. На рис. 3.2 представлены дерево наследования коллекторов и дерево наследования контекстов. В основе иерархии коллекторов лежит класс *CollectorBase*, в основе контекстов – *CollectorContext*.

Таблица 3.1

Поддерживаемые типы сборщиков

Наименование типа сборщика	Краткое описание	Цели сборки
PollerCollector	Активный коллектор, периодически собирающий информацию через равные промежутки времени	Абстрактный класс, не имеет цели
PingCollector	Проверяет доступность компьютеров за счет посылки ICMP Echo-Request	infrastructure.resources:Ping
HttpCollector	Сборщик данных по http-запросу	Абстрактный класс, не имеет цели
AsmxWebServiceCollector	Периодически вызывает операцию web-сервиса, основанного на технологии ASMX	Абстрактный класс, не имеет цели
ContractedWebServiceCollector	Периодически вызывает операцию web-сервиса, основанного на технологии WCF	Абстрактный класс, не имеет цели
InterpreterJobMonitor	Сборщик данных об исполняющихся композитных приложениях	jobs:Interpreter.Jobs
GridNNSResourcesCollector	Сборщик данных о ресурсах ГридННС	resources:GridNNS.Resources
IntegratorResourcesCollector	Сборщик данных о ресурсах сервиса «Метакластер»	resources:Integrator.Resources
HistoryCollector	Сервисный коллектор, который периодически переносит данные из актуального хранилища в хранилище истории	svc:Monitoring.ClearHistory

Для повышения устойчивости к сбоям компонента мониторинга менеджер сборщиков включает в себя механизм перезапуска коллекторов: если сборщик выходит из строя по какой-то причине, менеджер его перезапускает. При инициализации сборщика поддерживается опция запуска, которая указывает, использовать или нет данный механизм, а также указывает максимальное количество повторных запусков.

Рисунок 3.2 – Диаграмма классов сборщиков компонента мониторинга
а) дерево наследования сборщиков, б) дерево наследования их контекстов

Менеджер коллекторов ставит в соответствие каждому активному сборщику его состояние (рис. 3.3). Благодаря учету состояний менеджер может корректно управлять сборщиками. Алгоритм каждого коллектора отслеживает запрос на смену состояния и соответственно выполняет для этого определенные действия.

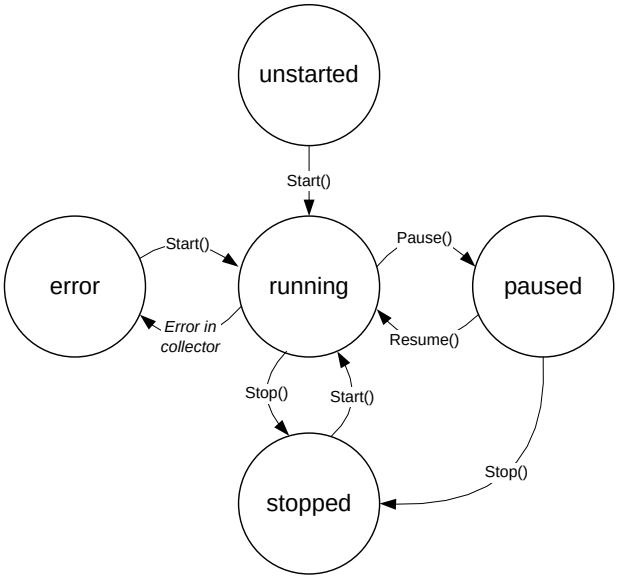

Рисунок 3.3 – Схема возможных состояний коллектора и переходов между ними в менеджере коллекторов

После получения данных от источника сборщик производит фильтрацию необходимой информации, а также агрегацию некоторых параметров. Далее он преобразует данные к унифицированному виду и сохраняет в хранилище информации об актуальном состоянии системы. При сохранении информации каждое поле структуры данных помечается временной отметкой поступления в хранилище, а также отметкой о том, когда эти данные перестанут быть актуальными. Значение последней отметки устанавливается субъективно разработчиком сборщика (например, оно может быть равно времени поступления в хранилище плюс интервал опроса). Данный механизм временных отметок необходим для вытеснения устаревших данных.

Для предварительного анализа поступающих в хранилище данных предусмотрен механизм анализаторов. Анализатор – это программа, которая идентифицирует высокоуровневые события из поступающей в хранилище актуальных данных информации и соответствующим образом обрабатывает их (например, оповещает администратора при недоступности ресурса).

Для возможности расширения функциональности модуля сбора данных предусмотрен механизм динамического добавления коллекторов. Дополнительные коллекторы можно добавлять в модуль в виде сборки .NET по технологии MEF (Managed Extensibility Framework) либо в виде скрипта на языке IronPython.

Для конфигурации модуля сбора данных и, в частности, менеджера коллекторов, а также механизма работы с хранилищами предусмотрен менеджер конфигурации (*Configurator*). Менеджер представлен в виде скрипта на языке IronPython.

Интерфейс КМ представляет собой web-сервис, призванный обеспечить доступ к функциональности компонента другим частям платформы. Он предоставляет набор вызовов для получения актуальной информации или истории изменений касательно ресурсов, входящих в платформу, исполняющихся композитных приложениях, произошедших в системе событиях.

3.2. Основные классы

Ниже приводятся сокращенные описания структуры и методов основных классов компонента мониторинга.

3.2.1. Класс CollectionManager

Менеджер сборки. Подгружает при старте динамические библиотеки с классами-сборщиками данных. Хранит в себе активные сборщики. Маршрутизирует запрос о запуске сборщика конкретному типу сборщиков.

Свойства

- CollectorPool (тип: CollectorPool) – пул сборщиков.
- CollectorLoader (тип: CollectorLoader) – загрузчик кода сборщиков из нескольких источников.

Методы

- void Collect (string targetType, CollectorContext context, CollectionOptions collectionOptions) – запрос на регистрацию нового сборщика информации.
  Аргументы: targetType – строка-идентификатор цели сборки, context – контекст сборщика, collectionOptions – опции сборщика.
- void Load () – динамическая загрузка библиотек кода, содержащих поддерживаемые сборщики данных.

3.2.2. Класс CollectorBase

Базовый класс для сборщиков данных. Исполняется в своем потоке. Поддерживает многопоточный доступ.

Свойства

- State (тип: CollectorState) – состояние коллектора;
- Context (тип: CollectorContext) – контекст коллектора ;
- Id (тип: Guid) – идентификатор коллектора.

События

- StateChangedEventHandler StateChanged – событие изменения состояния коллектора.

Методы

- void Collect() – функция сборки. Наследники класса реализуют данную функцию;
- void Initialize () – инициализация коллектора;
- void Pause () – пауза коллектора;
- void Restart () – перезапуск коллектора;
- void Resume () – возобновление коллектора;
- void Start () – запуск коллектора;
- void Stop () – остановка коллектора.

3.2.3. Класс CollectorLoader

Класс динамической загрузки кода сборщиков из нескольких источников. Сборщики являются плагинами и расширяют базовую функциональность компонента. Поддерживает загрузку сборщиков через механизм MEF, а также поддерживает сборщики, заданные на языке IronPython. Поддерживает многопоточный доступ.

Свойства

- ExtCollectorCreators (тип: IEnumerable< ICollectorCreator >) – точка входа для подсоединения MEF-плагинов.

Методы

- bool CanCreateCollectorByTargetType (string monitoringTargetType) – проверка, может ли данный класс создать коллектор, поддерживающий указанную цель.
  Аргументы: monitoringTargetType – цель сборки.
- bool CanCreateCollectorByType (string collectorType) – проверка, может ли данный класс создать экземпляр сборщика указанного типа.
  Аргументы: collectorType – строковое представление типа сборщика.
- CollectorBase Create (string collectorType, CollectorContext context) – создание экземпляра коллектора по его типу.
  Аргументы: collectorType – тип коллектора, context – контекст сборщика данных.
  Возвращает созданный экземпляр коллектора.
- CollectorBase CreateCollectorByTargetType (string monitoringTargetType, CollectorContext context) – создание коллектора по цели сборки.

Аргументы: monitoringTargetType – цель сборки, context – контекст коллектора. Возвращает созданный экземпляр коллектора.

- IList<string> GetAvailableCollectorTypes () – получение списка доступных типов сборщиков. Возвращает список из строковых представлений целей сборки.
- IList<string> GetAvailableTargetTypes () – получить список поддерживаемых целей сборки. Возвращает список из текстовых представлений целей сборки.
- void Load () – подключение модулей сборки.
- IEnumerable<ICollectorCreator> LoadExtentionCollectors () – подключение MEF-сборщиков. Возвращает список сборщиков.
- IEnumerable<ICollectorCreator> LoadPythonCollectors () – загрузка сборщиков заданных в Python-файлах. Возвращает список сборщиков.

Данные класса

- COLLECTORS_PATH (тип: string) – путь хранения кода сборщиков.

3.2.4. Класс CollectorPool

Пул коллекторов. Объект данного класса хранит список активных коллекторов. Отвечает за маршрутизацию запросов на сборку.

Методы

- bool CanMonitor (string monitoringTargetType) – проверка, содержится ли коллектор, который способен производить мониторинг ресурса.
  Аргументы: monitoringTargetType – тип ресурса. Возвращает результат проверки.
- IList<string> GetAvilableTargetTypes () – получить список поддерживаемых типов целей сборки. Возвращает список целей.
- void Monitor (string monitoringTargetType, CollectorContext context) – запрос на мониторинг ресурса. Маршрутизируется к конкретному коллектору.
  Аргументы: monitoringTargetType – тип цели мониторинга, context – контекст.
- void RunCollector (CollectorBase collector, CollectionOptions collectionOptions) – регистрация коллектора в пуле.
  Аргументы: collector – коллектор, collectionOptions – опции сборки.

3.2.5. Интерфейс IStorage

Интерфейс хранилища данных мониторинга. Включает методы сохранения и извлечения данных.

Свойства

- SupportHistory (тип: bool) – определяет, поддерживается ли история изменения информации.

Методы

- int CleanOldData (StoragePath from, TimeSpan savePeriod) – очистка устаревшей информации.

  Аргументы: from – путь до объектов хранения, savePeriod – время устаревания. Возвращает информацию, сколько объектов было очищено.
- BsonDocument GetActualEntityInBson (StoragePath storagePath) – получение актуальной информации, хранящейся по указанному пути.

  Аргументы: storagePath – путь до объекта хранения.
- string GetActualEntityInJson (StoragePath storagePath) – получение актуальной информации, удовлетворяющей запрошенному пути в хранилище мониторинга.

  Аргументы: storagePath – путь до объекта хранения.
- ыекштп хъ ПуеСщддусешщтТфьуы () – получить список коллекций. Возвращает список коллекций.
- string GetDataInJson (StoragePath storagePath) – получение данных по пути в виде Json.

  Аргументы: storagePath – путь до объекта хранения.
- T GetDataObject< T > (StoragePath storagePath) – получение объекта из хранилища (десериализация).

  Аргументы: T – тип объекта, storagePath – путь до объекта хранения.
- void SetDataObject< T > (StoragePath storagePath, T obj, TimeSpan actualFor) – сохранение объекта в хранилище.

  Аргументы: storagePath – путь до объекта хранения, obj – объект, actualFor – до какого момента данный объект будет актуальным, T – тип объекта.
- void SetFields< T > (StoragePath storagePath, IDictionary< string, T > fields, TimeSpan actualFor) – установить значения полей объекта хранения.

  Аргументы: T – тип поля, storagePath – путь до объекта хранения, fields – словарь полей, actualFor – до какого момента информация будет актуальной.

3.2.6. Интерфейс IMonitoringService

Контрактный интерфейс WCF-сервиса мониторинга. Реализован в классе MonitoringService.

Методы

- string GetActualDataInJson (string infoPath) – получение актуальной информации из хранилища мониторинга в формате Json.
  Аргументы: infoPath – путь до объекта хранения. Возвращает Json-представление запрашиваемой информации.
- JobDescription GetJobInfo (Guid jid) – получение информации о исполняющемся WF.
  Аргументы: jid – идентификатор WF.
- string GetJobInfoInJson (Guid jid) – получение информации об исполняющемся WF в формате JSON.
  Аргументы: jid – идентификатор WF.
- ResourceDescription GetResourcesInfo () – получение информации о состоянии ресурсов платформы Возвращает состояние ресурсов.
- string GetResourcesInfoInJson () – получение информации о ресурсе в формате JSON. Возвращает JSON-представление информации о ресурсе.

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Хранилища данных, входящие в состав КМ, реализованы на основе нереляционной СУБД MongoDB (с версией не ниже 1.6.5). Модуль сбора данных является обычным .NET-приложением. Для его функционирования требуется установленная платформа .NET версии не ниже 4.0. Интерфейс компонента мониторинга реализован в виде SOAP web-сервиса платформы .NET 4.0 на языке C#. Web-сервис разработан с использованием технологии WCF на основе стандарта SOAP. Хостинг сервиса производится в IIS (Internet Information Services). Компонент мониторинга предназначен для функционирования на аппаратных системах с характеристиками:

- архитектура процессора – x86, x86_64, IA64;
- минимальный объем оперативной памяти – 2 ГБ;
- минимальный объем свободного пространства на жестком диске – 20 ГБ;
- минимальная тактовая частота процессора – 2 ГГц.

Модуль требует для своей работы наличия следующего системного ПО: ОС семейства Windows NT (версии старше Windows 2000), .NET 4.0, IIS (с версией старше 6.0).

Необходимо отметить, что все модули, входящие в состав компонента, могут быть установлены на различные машины с тем условием, что будет обеспечен доступ по сети (протоколу TCP) к хранилищам данных (MongoDB).

5. ВЫЗОВ И ЗАГРУЗКА

Компонент мониторинга логически состоит из трех частей и подготовка его к работе заключается в настройке и запуске этих частей по отдельности в следующем порядке: хранилище данных, модуль сбора данных, интерфейс модуля.

Хранилища данных, входящие в состав компонента, могут находиться как в одной базе данных MongoDB, так и в разных. Желательно (но необязательно) установить MongoDB в виде сервиса операционной системы. Для работы компонента мониторинга предварительной настройки базы данных не требуется, кроме указания реквизитов доступа в конфигурационных файлах.

Модуль сбора данных является .NET-приложением и требует для своей загрузки только корректной конфигурации и запуска исполняемого файла «MonitoringDaemon.exe». Конфигурация модуля состоит в задании скрипта конфигурации «config.py», который написан на языке IronPython. В рамках данного скрипта можно настроить доступ к хранилищам, настроить и активировать коллекторы. Все расширения и дополнительные коллекторы должны быть размещены в директории «ExtCollectors». Они будут автоматически активированы при условии корректной настройки в скрипте конфигурации.

Интерфейс КМ реализован в виде SOAP WCF-сервиса платформы .NET. Для запуска сервиса используется стандартный механизм используемого web-сервера. Загрузка сервиса в этом случае выполняется web-сервером автоматически по мере поступления запросов от клиентов сервиса. КМ предоставляет интерфейс в виде WCF-сервиса, реализованного классом MonitoringService (см. спецификацию IMonitoringService в разделе 3.2.6). Вызов сервиса производится стандартным для технологии WCF способом: по опубликованному описанию сервиса (WSDL) необходимо создать прокси-класс, через который осуществляется взаимодействие с сервисом путем вызова необходимых операций (методов). Процедура создания прокси-класса зависит от того, на базе каких технологий строится клиентское приложение. В случае если выбран язык программирования C# и платформа .NET, построение клиента производится за счет

вызова служебного программного средства svcutil.exe, распространяемого в составе платформы .NET, либо за счет создания ссылки на сервис в среде Microsoft VisualStudio.

6. ВХОДНЫЕ ДАННЫЕ

Входными данными для компонента мониторинга являются конфигурационные файлы, а также входные параметры вызова интерфейсных методов. Интерфейс компонента конфигурируется файлом «Web.config», в котором указываются реквизиты доступа к базе данных, содержащей хранилище данных мониторинга (секция – StorageConnectionString). Реквизиты указываются в виде строки подключения в стандартном для MongoDB виде, например «mongodb://localhost/results». Модуль сборки конфигурируется скриптом «config.py» (см. пример на рис. 6.1). При старте модуля производится интерпретация данного скрипта, который осуществляет настройку системы. Функциональное назначение скрипта – настройка доступа к хранилищам, конфигурирование и активация коллекторов.

```
class PyConfigurator(IConfigurator):
    def ConfigureEnvironment(s):
        pass
    def ConfigureManagers(s,sm,cm):
        Log.Trace("Configuring storage manager..")
        ConnectionString = "mongodb://localhost/monitoring"
        sm.OpenConnection(ConnectionString);
        Log.Trace("Конфигурирование менеджера сборщиков")
        Log.Trace("Загрузка сборщиков")
        cm.Load()
        # =================== Коллекторы =========================
        # >> ping collector
        ctx = PingCollectorContext()
        ctx.Targets["192.168.1.16"] =
StoragePath(r"infrastructure.resources#{'&ID':'srv08'}")
        ctx.CollectionInterval = 30000
        ctx.Storage = sm.Storage
        cm.Collect("infrastructure.resources:Ping", ctx, CollectionOptions(True, 0))
        # >> integrator collector
        ctx = IntegratorResourcesCollectorContext()
        ctx.StaticInfoUpdateTimes = 20
        ctx.StorageBasePath = StoragePath("resources")
        ctx.Binding = System.ServiceModel.BasicHttpBinding()
        ctx.EndPointAddress =
System.ServiceModel.EndpointAddress("http://192.168.1.189/Integrator/IntegratorService
.asmx")
        ctx.CollectionInterval = 15000
        ctx.Storage = sm.Storage
        cm.Collect("resources:Integrator.Resources", ctx, CollectionOptions(True, 0))
        # >> interpreters job collector
        ctx = InterpreterJobMonitorContext()
        ctx.StorageBasePath = StoragePath("jobs")
        ctx.Binding = System.ServiceModel.BasicHttpBinding()
        ctx.Binding.MaxReceivedMessageSize = 655360
        ctx.Binding.MaxBufferSize = 655360
        ctx.EndPointAddress =
System.ServiceModel.EndpointAddress("http://localhost/Workflow/FlowSystemService.svc/F
lowSystemService")
        ctx.CollectionInterval = 5000
        ctx.Storage = sm.Storage
        cm.Collect("jobs:Interpreter.Jobs", ctx, CollectionOptions(True, 0))
configurator = PyConfigurator()
```

Рисунок 6.1 – Пример файла конфигурации модуля сборки

7. ВЫХОДНЫЕ ДАННЫЕ

Выходными данными КМ являются результаты вызовов интерфейсных операций, которые возвращают запрашиваемые данные в формате JSON или в виде сериализованной объектной структуры в формате SOAP. Структурирование выдаваемой информации в виде SOAP необходимо для закрепления схемы взаимодействия с другими компонентами комплекса. К таким операциям относятся: операция получения информации о выполняемых или выполненных композитных приложениях (GetJobInfo), операция получения информации о вычислительных ресурсах комплекса (GetResourcesInfo). Для остальных операций в интерфейсе используется формат JSON. На рис. 7.1 приведен пример возвращаемой информации о вычислительном ресурсе в виде JSON-документа.

```
{
  "&SOURCE": {
    "format": "Integrator",
    "collector": "IntegratorResourcesCollector",
    "endpoint": "http://192.168.1.189/Integrator/IntegratorService.asmx"
  },
  "&ID": "cluster_niinkt_1",
  "name": "cluster_niinkt_1",
  "nodeCount": 2,
  "total_load": 3,
  "total_core_count": 16,
  "total_free_hdd_size": 140.638,
  "total_RAM_size": 32168.078125,
  "nodes":[
      { "name": "i-master.nanocomputer.net",
        "RAM_size": 16084.0390625,
        "number_of_cores": 8,
        "software_packages":["ORCA","GAMESS","SEMP","PLASMON","DALTON","NTDMFT","MD-
KMC",
"PRIRODA","NAEN","MAGNET","UPCONVERSION","TESTP","BELMAN","OpenFOAM","NANOFLOW"],
        "core_frequency": 2666000,
        "load": 6,
        "free_hdd_size": 67.635,
        "free_hdd_percent": 53.326868036994114
      },{
        "name": "i-node.nanocomputer.net",
        "RAM_size": 16084.0390625,
        "number_of_cores": 8,
        "software_packages":["ORCA","GAMESS","SEMP","PLASMON","DALTON","NTDMFT","MD-
KMC",
"PRIRODA","NAEN","MAGNET","UPCONVERSION","TESTP","BELMAN","OpenFOAM","NANOFLOW"],
        "core_frequency": 2666000,
        "load": 0,
        "free_hdd_size": 73.003,
        "free_hdd_percent": 57.559271786865985
      }],
  "tasks": [],
}
```


Рисунок 7.1 – Пример выдаваемой модулем информации о состоянии ресурса в формате JSON

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

JSON	JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript)
MEF	Managed Extensibility Framework
SOAP	Simple Object Access Protocol (простой протокол доступа к объектам)
WCF	Windows Communication Foundation (программный фреймворк)
WF	Workflow (поток заданий)
БД	База данных
КМ	Компонент мониторинга
МИТП	Многопрофильная инструментально-технологическая платформа
ОС	Операционная система
ПО	Программное обеспечение

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	изменённых	заменённых	новых	Аннулированных					