СУ “Св. Климент Охридски”
Факултет по Математика и Информатика
Катедра “Информационни технологии”

Дипломна работа

Тема:
Адаптируем SOAP базиран клиент за управление диагностика и контрол на вградени измерителни устройства

Дипломант: Христина Костова Кънчева
Специалност: Разпределени системи и мобилни технологии
Факултетен номер: M21825

Научен ръководител: Гл. Ас. Д-р Михаил Аврамов
Консултант: Гл. Ас. Елиза Стефанова

София, 2007

Съдържание:

Въведение

С все по-развиващите се информационни технологии в наши дни ролята на вградените устройства в повечето системи е основна и значима. Това води и до необходимост за съвместно управление, диагностика и контрол на тези устройства. За нарастващите нужди на съвременните технологии извършването на такива операции често налага използването на протоколи от високо ниво, а изискванията на потребителите не се покриват с предоставяната функционалност на WEB базираните интерфейси.

Повечето сложни системи от вградени устройства имат необходимост от клиент, позволяващ следенето, конфигурирането и управлението им. Разбратката на такива специфични за всяка система клиенти е скъпо и неефективно.

Настоящата дипломна работа предоставя едно важно решение на този проблем чрез разработка на универсален клиент за достъп до функциите и за диганостика на една такава система. За постигане на тази задача е използван абстрактен протокол, който не зависи от функционлността на системата. Реализацията на такъв абстрактен протокол е осъществена на базата на SOAP протокол с използване на AXIS toolkit и gSoap. Задачите за осъществяването на универсалния клиент са следните:

- Разработване на SOAP протокол за управление на вградени устройствата.
- Проектиране на клиентското приложение
- Реализация на клиентското приложение
- Интеграция на клиента за работа с камера за управление на заваръчен робот на фирмата KDS.
- Интеграция на клиента за работа със система за микроизмервания.

Стъпките при разработката и решението на проблема и задачите са следните:

1. SOAP протокол, AXIS toolkit и gSOAP toolkit – запознаване със SOAP, AXIS и gSOAP; какво представляват SOAP протокола и AXIS toolkit-а; какви средства предоставят за разработка; при какви проблеми са подходящи за използване; основни цели и похвати на SOAP протокола, преимущества и недостатъци на SOAP.
2. Описание на протокола за комуникация – как е реализиран комуникационния протокол – технология; какви методи и функции предоставя протокола за

управление, диганостика и контрол на системата от вградени устройства; специфични характеристика на системата и протокола - канали за комуникация, транзакции, събития; описания на интерфейса на протокола.

3. Архитектура на клиента – организация на клиента и основни модули за функциите на клиентското приложение. Каква функция изпълнява всеки един от модулите и за какво служат те.
4. Структура на клиента – какво предоставя клиентското приложение на потребителя, как е организирана и разположена информацията в клиента, какво и как се визуализира в клиента, какво е предоставено за работа на потребителя, защитен достъпо до данните на системата, реализация на клиентеското приложение, команди към систеамта.
5. Примерни резултати от работата на приложението - работа с камерата за управление на заваръчен робот на фирмата KDS, описание на свойствата на камерата, които системата поддържа – свойства за алармиране, свойства на каналите, режим на работа(администраторски и потребителски), свойства на сензоритеи др. Специални свойства на камерата – снимки, профили, заваръчна точка, агрегиращи линии и др. Работа със система за микроизмервания разработвана в Химически факултет на СУ “Св. Кл. Охридски”.

Глава 1 – SOAP и Apache AXIS

1.1. Основни характеристики на SOAP

SOAP е прост XML-базиран протокол, позволяващ на приложенията да обменят информация (съобщения) помежду си, обикновено през HTTP/HTTPS. Основните характеристика на протокола са [1]:

- SOAP е съкращение на Simple Object Access Protocol
- SOAP е протокол за комуникация
- SOAP е протокол за достъп до Web Service
- SOAP е за комуникация между приложения
- SOAP е формат за изпращане на съобщения
- SOAP е създаден за комуникация в интернет
- SOAP е платформено независим
- SOAP е езиково независим
- SOAP е базиран на XML съобщения
- SOAP е прост и разширяем
- SOAP позволява преминаването през защитните стени (firewalls)
- SOAP е разработен и като W3C стандарт

Важно предизвикателство в днешно време за разработчиците на приложения е да осигурят интернет комуникация между програмите. Един от разпространените методи за сега е употребата на отдалеченото викане на процедура (RPC) между обектите, при което клиентът изпраща искане (request message) към сървъра и сървъра веднага изпраща отговор на клиента. Употребата на RPC води до проблеми със съвместимостта и сигурността. Защитните стени и прокси сървърите обикновено блокират и не позволяват този вид трафик. По-добър начин за комуникация между приложенията е през HTTP, защото HTTP се поддържа от всички сървъри. SOAP е създаден да реализира именно такъв тип комуникация. SOAP дава начин за комуникация между приложения, работещи на различни операционни системи, с различни технологии и програмни езици. SOAP използва протоколите на интернет приложното ниво като транспортни протоколи. SMTP и HTTP са типични приложни протоколи, използвани като транспортни в SOAP, но HTTP е предпочитания, поради добрата му функционалност в днешната интернет инфраструктура. SOAP може да се използва още и на базата на HTTPS протокола с проста или взаимна аутентикация (предпочитания

метода за осигуряване на защита при WEB услугите). Това е едно от основните предимства на SOAP протокола.

Друг разпространен метод за комуникация е чрез използването на CORBA. CORBA (Common Object Request Broker Architecture) е стандарт, който позволява, софтуерни компоненти, написани на различни езици и работещи на няколко компютъра, да комуникират помежду си и да работят заедно [2]. Протоколът SOAP в сравнение с CORBA е по-ограничен. Например той има ограничения при поддръжката на обектно ориентираните концепции като наследяване или при работа с транзакции. В сравнение обаче с тези ограничения, SOAP е много по-прост. Благодарение на това, че SOAP използва HTTP протокола, дефакто SOAP сървърите са Web сървъри. Това, че SOAP протокола е разработен на основата на HTTP протокола и на XML го прави лесен и удобен за използване. Тези характеристики правят SOAP протокола изключително подходящ за приложения, които изискват отдалечени викания на процедура и не зависят от обектно ориентираната концепция на наследяването и обработката на транзакции. При такива приложения използването на CORBA само би утежнило и увеличило производителността [3].

Описаните до момента характеристики за SOAP показват, че SOAP протокола оформя основното ниво на множеството от WEB услуги. Това се подсигурява още и благодарение на осигурената от SOAP основна рамка на съобщенията, на която по-абстрактните нива могат да разчитат. SOAP е наследника на XML-RPC, въпреки че заимствува транспортирането, неутрално взаимодействие и envelope/header/body структурата от други места, най-вероятно от WDDX. За стандартен формат на съобщенията е избран XML поради неговата широка употреба и усилията в „open source” напредъка.

SOAP съобщението представлява обичен XML документ съдържащ следните елемент[4]:

- задължителен envelope елемент, който идентифицира XML документа като SOAP съобщение
- незадължителен header елемент, който съдържа основна информация
- задължителен body елемент, който съдържа информация за виканите услуги и отговорите
- незадължителен fault елемент, който предоставя информация за грешки, възникнали по време на обработката на съобщението

Всички изброени елементи са декларирани в основното пространство от имена на SOAP протокола.

Малко дългия синтаксис на XML може да бъде и полза и пречка. Форматът му е лесен за разчитане от човека, но може да е сложен, което води до увеличаване на времето на изпълнение. От друга страна апаратурата ускорява обработката на XML съобщенията.

1.2. Преимущества на SOAP протокола

- изплезването на SOAP през HTTP позволява по-лесната комуникация през прокси сървъри и защитни стени в сравнение с познатите до сега технологии за отдалечени изпълнения.
- SOAP е достатъчно гъвкав за употребата на различни транспортни протоколи. Стандартно се използва HTTP протокола като транспортен протокол, но други протоколи (TCP, SNMP) също са приложими.

1.3. Недостатъци на SOAP протокола

- поради формата на XML, SOAP може да бъде по-бавен в сравнение с някои middleware технологии
- при използването на HTTP като транспортен протокол ролите на участващите страни са фиксирани. Само едната страна (клиентът) може да използва услугите на другата (сървъра)
- много от SOAP имплементациите ограничават размера на данните, който може да се изпращат
- повечето употреби на HTTP протокола като транспортен протокол се правят без ясна представа как точно дадена операция би била моделирана в HTTP. Поради това няма добър начин да се разбере дали дадения метод е най-подходящия за съответната операция.

1.4. Какво е Apache AXIS?

Apache AXIS представлява Web Services базиран на SOAP. Състои се от Java и C++ имплементация на SOAP сървъра. Предоставя различни средства и API за генериране на Web Service приложения. Apache AXIS предоставя следните възможности за разработчиците [5]:

1.4.1. Програмни средства

- API за XML-базирани клиенти, включително поддръжка на WSDL
- поддръжка на POJO и Spring сървиси и клиенти
- поддръжка на всякакъв вид обмяна на съобщения
- синхронни и асинхронни викания
- предоставя модел за архивирано деплойване на service, поддържащ пълна енкапсулация на service-а с поддръжка на версии
- предоставя модел за архивирано деплойване на модули, поддържащ контролирано разширяване със поддръжка на версии
- деплойване на Web services или отделни части докато системата е стартирана и работи. Това позволява нови услуги да бъдат добавени към системата без да се налага тя да бъде спряна
- генериране на допълнителен WS-Policy код
- гъвкав модел на жизнения цикъл на услугите
- автоматична поддръжка на POX (REST) стила за обръщение към услуги
- поддръжка на заявка към: WSDL услуга (чрез ?wsdl); схема (чрез ?xsd); policies (чрез ?policy)
- WSDL
- анотация на POJO
- интеграция на JAX-WS
- клиентско деплойване
- двоична сериализация
- поддръжка на JSON
- поддръжка на EJB Provider

1.4.2. Спецификации, които поддържа

- SOAP 1.1 и 1.2
- Message Transmission Optimization Mechanism (MTOM), XML Optimized Packaging (XOP) и SOAP with Attachments
- WSDL 1.1, включително и за SOAP и за HTTP
- WS-Addressing (за предоставяне и крайна)
- WS-Policy
- SAAJ 1.1

1.4.3. Транспортни протоколи

- HTTP
- SMTP
- JMS
- TCP

1.4.4. Поддръжка на данни

- Axis Data Binding (ADB)
- XMLBeans
- JibX
- JaxMe (Experimental)
- JaxBRI (Experimental)

Глава 2 – WEB Service - Протокол за комуникация

2.1. Основни характеристики на устройството управлявано през предлагания интерфейс

Устройството предлага колекция от йерархично организирани свойства които имат ТИП, СТОЙНОСТ и др. параметри (пояснение, мерни единици, допустими стойности и др.). Функциите на устройството се управляват изцяло с помощта на въпросните свойства. Свойствата могат да бъдат прочитани и записвани. Промяната на стойността на свойствата може да доведе до промяна в параметрите на работа на устройството или да предизвика директни действия (/sensor/LASER_ON=1 включва лазера). Някой свойства са само за четене. Чрез тях устройството предоставя диагностична информация или резултати. За поддръжка на стрийм информация (напр. снимка с обработка и резултати от измерване) протоколът предлага функция за избор на пакет. След избиране на пакета отделните елементи в него могат да бъдат прочетени последователно.

2.1.1. Технологии

Протоколът е базиран на SOAP. Той предлага малко множество от методи, които осигуряват достъп и контрол до всички функции на системата.

Целта на разработката е да осигури контрол върху операциите на системата от вградени устройства.

Контролът се осъществява чрез промяна на контролни параметри (наречени свойства) на системата. Някои свойства са конфигурационни а при промяна на други устройството изпълнява специфични функции. Напр. /sensor/LASER_ON=1 предизвиква включване на лазера на камерата.

2.1.2. Преглед на функциите

За да се управляват ефективно параметрите на устройството е необходимо оператора да получава адекватна информация за резултатите от обработката на изображения извършвани в системата. При това той е в състояние да се намеси ефективно и да настрои параметрите на използваните алгоритми така че да получава оптимални резултати.

За определяне кои параметри да бъдат променени, протоколът осигурява възможността да бъдат следени резултатите по време на изпълнение на системата. В зависимост от конфигурацията на устройството той предоставя различни междинни резултати – снимки, извлечени точки, линии, коригиран шаблон и други.

Свойствата имат описание, което може да бъде извлечено чрез getPropertyDefinition метода.

Имената на свойствата са йерархични и включват префикс, който предоставя възможността за логическо групиране на свойствата. Например
/sensor/ директория обхваща свойства, отнасящи се за CMOS сензора – прозорец, integration time и т.н.

2.1.3. Транзакции

Някои свойства на системата могат да бъдат добавени в транзакция. Транзакция може да бъде дефинира в ниво на директория като например свойствата в **/processing/object_geometry/** са част от транзакция. Транзакцията е автоматичен начин да се промени стойността в системата на напълно променени множества от свойства наведнъж (в един момент от времето). За всички транзакционни директории потребителския интерфейс би трябвало да поддържа функция (бутони) за изпълнение на транзакция и функция (бутони) за изчистване на транзакция. Имплементацията поддържа автоматично beginTransaction за първото присвояване на стойност на свойство в транзакционна директория. След това всички свойства от същата директория променени от потребителския интерфейс се кешират на сървъра (но некоригирани все още в системата. Викането на *commitPropertyTransaction*от потребителския интерфейс променя всички свойства в системата по синхронен - последователен начин.

2.1.4. Канали

Данни получени от системата по време на обработка могат да бъдат използвани от клиента чрез референта схема за записи и снимки вътре в запис чрез *selectPacket* функцията. Канала е поток с информация за данните от обработката, който може да бъде „заключен” и всяка част от тази информация може да бъде извлечена с подходящите getProperty… методи. Някои канали имат фиксирано съдържание, което може да бъде променено само в различните издания на системата, но в рамките на една

версия са статични. Някои други канали могат да имат програмно (конфигурируемо) съдържание.

2.1.5. Събития

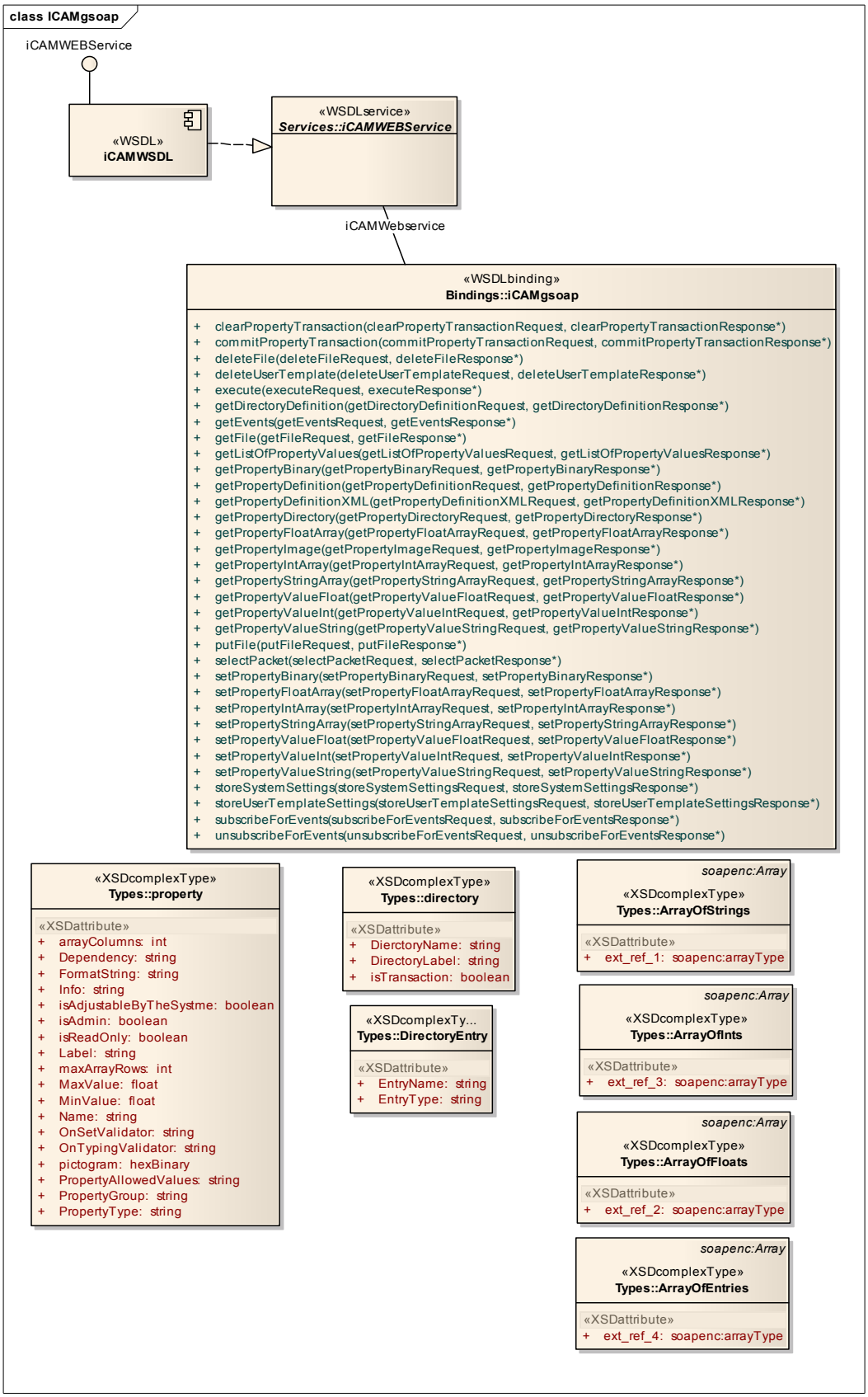
Механизмът поддържа уведомяване за параметри модифицирани от системата. За извличане на уведомение за променен параметър на канал клиентът трябва да се запише за директорията където това събитие се намира. Всяка промяна от началното ниво надолу по директория ще бъде изпратено към клиента като събитие (клиентът получава периодично събития, за които се с записал). Ако структурата на директорията, за която клиентът се е записал се промени, съдържанието на събитието е самата директория. Когато клиентът извлече списъка със събитията (имената на променените свойства) той трябва да поиска стойностите им или да презареди структурата и след това да поиска стойностите.

2.2. Интерфейс на протокола

Протокол: SOAP

Стил по подразбиране: rpc

Транспортиране: SOAP на базата на HTTP



Диаграма 1. Интерфейс на протокола за комуникация

2.2.1. Операции

getPropertyDirectory

Описание: Функцията извлича списък от имена на свойства на дадено ниво от йерархията на свойствата. Например: името на нивото (директорен параметър)“ // за кореновото ниво, /sensors/ за сензор нивото. Забележка: за извличане на свойства трябва да се даде като параметър нивото на което те принадлежат / (/sensor/ за сензор нивото или // за кореновото ниво) за извличане на поднива трябва да се въведат само водещите наклонени черти (/ - подниво на кореновото ниво, /sensor подниво на сензор нивото).

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyDirectoryRequest (soap:body, use = literal)

Directory тип *string*

Изход: getPropertyDirectoryResponse (soap:body, use = literal)

DirectoryContent тип *ArrayOfEntries* – стринг от тип *DirectoryEntry*

error тип *string*

getDirectoryDefinition

Описание: Този метод връща структура с директорийни специфични дефиниции. Важно поле в тази структура е isTransaction. В случай, че полето е истина промяната на стойността на всички свойства в тази директория се осъществява с транзакция

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход:

directoryName тип *string*

Изход: getDirectoryDefinitionResponse (soap:body, use = literal)

DirectoryDefinition тип *directory*

- DierctoryName тип *string*
- DirectoryLabel тип *string*
- isTransaction тип *boolean*

error тип *string*

getPropertyDefinition

Описание: Връща информация за дефиницията на свойство. Структурата на "property" дефиницията предоставя детайлите

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyDefinitionRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyDefinitionResponse (soap:body, use = literal)

PropertyDefinition тип *property*

Структура, използвана за връщане на дефиницията на свойство. Клиентът трябва да използва дефиницията на свойствата за извличане на мета данни за свойствата предоставени в системата. В комбинация с метода getPropertyDirectory методът getPropertyDefinition позволява установяване на browsing имплементация на клиента.

- Name тип *string*

Името на свойство. Трябва да съдържа някаква йерархия например /sensor/exposuretime

- Label тип *string*

Наименование на това свойство, което ще бъде показано в ГПИ.

- OnTypingValidator тип *string*

On typing regular expression validator. Валидатора трябва да бъде прилаган за всеки набран символ в полето за въвеждане. Ако резултатът от проверката на валидатора за въведения текущ символен низ в полето е истина, то последната промяна се приема за правилна, иначе последната промяна се приема за неправилна и се извършва "rolledBack" връщане на транзакцията от клиента към последната валидна ситуация.

- OnSetValidator тип *string*

On set regular expression validator. Валидаторът трябва да съдържа по-прецизна проверка за валидност, която да бъде приложима когато цялата информация в полето е налична (валидаторът трябва да бъде приложен в случай на изход от полето). Ако резултатът от проверката на валидатора за въведения израз не е

истина, то съдържанието на полето не е вярно и клиентът изкарва съобщение за грешка.

- PropertyType тип *string*

Дефиницията тук поддържа подмножество на типовете приложими за системата и няма главни целеви комбинации, които не са възможни за устройството. За всички типове изброени по-долу има отделни функции за извличане и присвояване на стойностите им, които позволяват операция без typecasting и изрични превръщания в клиента. Може да бъде: intArray – списък от 32битови знакови integer стойност int - 32битова знакова integer стойност float – стойности с плаваща запетая floatArray списък от стойности с плаваща запетая string – стрингова стойност stringArray – списък от стрингови стойности enumeration – свойство, което приема специфично множество от стрингови стойности. Свойството PropertyAllowedValues съдържа списък от стрингове от допустими стойности dynamiclist – подобно на enumeration възможните стойности за enumeration се взимат от друго stringArray свойство. stringArray може да бъде извлечено чрез заявка на свойството, определено в PropertyAllowedValues. image – свойство съдържащо RLE кодиране на BMP снимки - blob – свойство съдържащо стойност от тип специфичен за клиента (например pictogram-ите са снимки съхранени на сървъра, които ще бъдат представени в клиента, интерпретиран от приложението)

- PropertyGroup тип *string*

Групата на свойство може да бъде шаблонна или системна. В зависимост от групата, към която свойството принадлежи то се съхранява съответно със storestoreJobSettings или storeSystemSettings методи.

- isReadOnly тип *boolean*

в случай на истина свойството е само за четене. Тази дефиниция може да зависи от администраторски режим, сменян в клиента.

- isAdmin тип *boolean*

в случай на истина свойството е само за административен достъп. То не трябва да бъде показвано в нормалния потребителски интерфейс.

- Info тип *string*

Съдържа допълнителна информация, която може да бъде използвана за визуализирането на свойството. Например типа на управлението, което се използва за визуализация (combo, checkbox, list и т.н.)

- PropertyAllowedValues тип *string*

Полето съдържа списък от стрингове, съдържащи допустими за свойството стойности. Желателно е да бъде дефинирано за така наречените изброими свойства. Свойството съдържа също така и указател към свойство от което да извлече разрешените свойства в случай на тип на свойство dynamiclist.

- MinValue тип *float*

Минимална стойност на свойството. В случай, че свойството е ArroayOfInts, ArrayOfFloats, int или float. В други случаи това поле не трябва да бъде вземано в предвид.

- MaxValue тип *float*

Максимална стойност на свойството. В случай, че свойството е ArroayOfInts, ArrayOfFloats, int или float. В други случаи това поле не трябва да бъде вземано в предвид.

- Dependency тип *string*

Зависимост от свойството - съдържа списък от свойства, които зависят от това свойство. В случай, че това свойство се промени този списък може да стане невалиден и трябва да бъде презареден от сървъра.

- isAdjustableByTheSystme тип *boolean*

Свойства, които могат да бъдат регулирани от системата трябва да бъдат прочетени след присвояване, за да бъде извлечено действителната стойност. например присвояване на стойност на свойство /sensor/exposuretime със стойност от ГПИ не може да бъде направена с абсолютна точност съответна на хардуера на сензора. Системата ще изчисли най-близката възможна стойност и свойството ще я върне (при поискване) като резултат от викането на setPropertyValueFloat("/sensor/exposuretime", some value). За свойства, които се изчисляват от системата като коефициент на успеваемост и т.н., това свойство е истина но isReadOnly би било също истина.

- FormatString тип *string*

Съдържа C формат string за изход

- pictogram тип *hexBinary*
- arrayColumns тип *int*

Атрибутът дефинира номера на колоните, които списъкът притежава. getProperty.... функцията връща едномерен масив, който може да бъде интерпретиран в клиента като двумерен.

- maxArrayRows тип *int*

Дефинира максималния размера на списъка.

error тип *string*

getPropertyDefinitionXML

Описание: Извлича описанието на свойство. Описанието е в XML кодиран файл, който съдържа данни за свойството (минимум какво е налице в структурата на свойството и евентуално повече допълнителна информация).

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyDefinitionXMLRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyDefinitionXMLResponse (soap:body, use = literal)

PropertyDefinitionXML тип *base64Binary*

error тип *string*

getPropertyFloatArray

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyFloatArrayRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyFloatArrayResponse (soap:body, use = literal)

floatArray тип *ArrayOfFloats* – списък от тип *float*

error тип *string*

getPropertyImage

Описание: Връща снимка от текущия пакет с данни. Имена на снимката могат да бъдат: за да се види дали пакетът съдържа специфична снимка трябва да се използва getPropertyDefinition с параметър подходящия канал и запис.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyImageRequest (soap:body, use = literal)

channel тип *string*

PropertyName тип *string*

Изход: getPropertyImageResponse (soap:body, use = literal)

Image тип *base64Binary*

error тип *string*

getPropertyIntArray

Описание: Връща 32 битов списък от integer стойности

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyIntArrayRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyIntArrayResponse (soap:body, use = literal)

int32Array тип *ArrayOfInts* – списък от тип *int*

error тип *string*

getPropertyStringArray

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyStringArrayRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyStringArrayResponse (soap:body, use = literal)

stringArray тип *ArrayOfStrings* – списък от тип *string*

error тип *string*

getPropertyValueFloat

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyValueFloatRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyValueFloatResponse (soap:body, use = literal)

floatValue тип *float*

error тип *string*

getPropertyValueInt

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyValueIntRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyValueIntResponse (soap:body, use = literal)

intValue тип *int*

error тип *string*

getPropertyValueString

Описание: Връща стойността на свойството зададено чрез името си

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyValueStringRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyValueStringResponse (soap:body, use = literal)

PropertyValue тип *string*

error тип *string*

getPropertyBinary

Описание: Извлича текущия пакет от канал, определен чрез името си. Операцията е атомична и не е влияеща

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getPropertyBinaryRequest (soap:body, use = literal)

PropertyName тип *string*

Изход: getPropertyBinaryResponse (soap:body, use = literal)

Data тип *base64Binary*

error тип *string*

setPropertyBinary

Описание: Слага пакет с данни в зададения канал. Процедурата може да бъде използвана за повторна обработка.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyBinaryRequest (soap:body, use = literal)

PropertyName тип *string*

Data тип *base64Binary*

Изход: setPropertyBinaryResponse (soap:body, use = literal)

error тип *string*

setPropertyFloatArray

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyFloatArrayRequest (soap:body, use = literal)

PropertyName тип *string*

floatArray тип *ArrayOfFloats* – списък от тип *float*

Изход: setPropertyFloatArrayResponse (soap:body, use = literal)

error тип *string*

setPropertyIntArray

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyIntArrayRequest (soap:body, use = literal)

PropertyName тип *string*

int32Array тип *ArrayOfInts* – списък от тип *int*

Изход: setPropertyIntArrayResponse (soap:body, use = literal)

error тип *string*

setPropertyStringArray

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyStringArrayRequest (soap:body, use = literal)

PropertyName тип *string*

stringArray тип *ArrayOfStrings* – списък от тип *string*

Изход: setPropertyStringArrayResponse (soap:body, use = literal)

error тип *string*

setPropertyValueFloat

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyValueFloatRequest (soap:body, use = literal)

PropertyName тип *string*

floatValue тип *float*

Изход: setPropertyValueFloatResponse (soap:body, use = literal)

error тип *string*

setPropertyValueInt

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyValueIntRequest (soap:body, use = literal)

PropertyName тип *string*

intValue тип *int*

Изход: setPropertyValueIntResponse (soap:body, use = literal)

error тип *string*

setPropertyValueString

Описание: Присвоява стойност на свойство зададено чрез името си

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: setPropertyValueStringRequest (soap:body, use = literal)

PropertyName тип *string*

PropertyValue тип *string*

Изход: setPropertyValueStringResponse (soap:body, use = literal)

error тип *string*

selectPacket

Описание: Командата за извличане на пакет се използва за извличане на пакет от запазено множество от пакет. По време на операцията са поддържани следните канали: /channels/video/ /channels/realtime/ /channels/debug/ /channels/statistics/ - Заявка за съдържанието на канал може да бъде направена чрез getPropertyDirectory функцията, която показва реалните налични свойства в канала. selectPacket функцията преглежда съдържанието на дадения канал с указания кадър и след това има възможност да извлече стойностите на различни свойства без реална несъвместимост по време на работа на системата. Опит за извличане на свойство от поддиректория на канала /channels/ без да е била извикана преди това функцията selectPacket ще доведе до грешка. В текущата реализация (session less) selectPacket работи само върху един канал. Избиране на друг канал освобождава избирането на първия канал. За реализацията на изискването за съхранение само на записи (информация между startTrack/endTrack командите викани от ROBOT в сериния интерфейс) се въвежда друг индекс на кадрите – запис (track). Старите записи имат номер започващ от 1 когато системата работи. Винаги има текущ запис с номер 0, което е валиден дори извън startTrack/endTrack последователността и може да бъде използван за текущ for current наблюдение на обработката. Различните записи се индексират чрез поддиректория на канал: /channels/video/0/.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: selectPacketRequest (soap:body, use = literal)

channel тип *string*

indexType тип *string*

Index тип *int*

Изход: selectPacketResponse (soap:body, use = literal)

FrameCounter тип *int*

error тип *string*

subscribeForEvents

Описание: Записването за събития е планирано, така че да дефинира кои събития да бъдат пратени на клиента. В първото изпълнение записването за събития няма да има истинска реализация.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: subscribeForEventsRequest (soap:body, use = literal)

listOfEventsToSubscribeFor тип *ArrayOfStrings* – списък от тип *string*

Изход: subscribeForEventsResponse (soap:body, use = literal)

error тип *string*

unsubscribeForEvents

Описание: Отписване от събития, за които е имало записване преди това. Отписването трябва да връща предупреждение (грешка), ако се прави опит за отписване от събитие за което е нямало записване, но трябва да се извърши отписване за останалите събития подадени в списъка, за който има записване. Функцията ще бъде реализирана заедно със subscribeForEvent

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: unsubscribeForEventsRequest (soap:body, use = literal)

listOfEventsToUnSubscribeFor тип *ArrayOfStrings* – списък от тип *string*

Изход: unsubscribeForEventsResponse (soap:body, use = literal)

error тип *string*

getEvents

Описание: Методът извлича списък от всички асинхронни съобщения в системата. Например смяна на свойства в системата. Събитието съдържа името на модифицираното свойство. Методът е предназначен да бъде викан на някакъв постоянен период от клиентското приложени (например на определен таймер).

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getEventsRequest (soap:body, use = literal)

Изход: clearPropertyTransactionResponse (soap:body, use = literal)

events тип *ArrayOfStrings* – списък от тип *string*

error тип *string*

getListOfPropertyValues

Описание: Методът връща списък от стойности на свойства базиран на списък от имена на свойства. Резултатът съдържа стойности подредени в реда на списъка при повикване. Всички стойности са форматирани като стрингове.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getListOfPropertyValuesRequest (soap:body, use = literal)

listOfProperties тип *ArrayOfStrings* – списък от тип *string*

Изход: getListOfPropertyValuesResponse (soap:body, use = literal)

listOfValues тип *ArrayOfStrings* – списък от тип *string*

error тип *string*

clearPropertyTransaction

Описание: Изчиства променените свойства до времето на последната commitPropertyTransaction

Тип на операцията: *Request-response.* Крайната точка получава съобщение и съответно изпраща подходящо съобщение.

Вход: clearPropertyTransactionRequest (soap:body, use = literal)

directoryName тип *string*

Изход: clearPropertyTransactionResponse (soap:body, use = literal)

error тип *string*

commitPropertyTransaction

Описание: За свойства, които принадлежат на транзакционна директория (само директна транзакция е приложима – без наследяване) този метод прилага транзакцията

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: commitPropertyTransactionRequest (soap:body, use = literal)

directoryName тип *string*

Изход: commitPropertyTransactionResponse (soap:body, use = literal)

error тип *string*

storeUserTemplateSettings

Описание: Съхранява свойството трайно, така че то е налично и при следващото стартиране на системата. Свойството, което се съхранява се задава с име. Ако името е ниво (например /sensor/control/) всички свойства на това ниво, които могат да бъдат съхранени (виж дефиницията на структурата на свойствата) ще бъдат съхранени.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: storeUserTemplateSettingsRequest (soap:body, use = literal)

templateKey тип *string*

Изход: storeUserTemplateSettingsResponse (soap:body, use = literal)

error тип *string*

storeSystemSettings

Описание: Съхранява системните настройки. В сравнение с storeJobSetttings тук има само един runtime и постоянно хранилище за тези свойства.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: storeSystemSettingsRequest (soap:body, use = literal)

Изход: storeSystemSettingsResponse (soap:body, use = literal)

error тип *string*

deleteUserTemplate

Описание: Изтрива шаблона.

Вход: deleteUserTemplateRequest

templateKey тип *string*

Изход: deleteUserTemplateResponse

error тип *string*

execute

Описание: Изпълнява команда (ОС ниво) на командния ред на сървъра. Изпълнението се контролира от executionOption параметър. Кореновата директория за операцията е глобален системен параметър който не е достъпен за нормална потребителска операция. За сигурност на механизъма постоянният корен ще бъде запазен в папка в ramfs. Всяка файлова операция в този случай ще се осъществява в ramFS. Изпълняваните операции може да имат постоянни резултати само чрез използването на администраторския режим при операция of operation, която може да предостави различни корени или чрез изпълнението на скрипт, който прави необходимото копие във Flash паметта.

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: executeRequest (soap:body, use = literal)

CommandLine тип *string*

ExecutionTimeout тип *int*

Изход: executeResponse (soap:body, use = literal)

error тип *string*

deleteFile

Описание: Изтрива упоменатия файл. Коренът на файла е дефиниран чрез глобални настройки на механизма

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: deleteFileRequest (soap:body, use = literal)

fileToDelete тип *string*

Изход: deleteFileResponse (soap:body, use = literal)

error тип *string*

getFile

Описание: Извлича съдържанието на файл. Корена на операцията е дефиниран в глобалните настройки (не е достъпен за протокола).

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: getFileRequest (soap:body, use = literal)

FileName тип *string*

Изход: getFileResponse (soap:body, use = literal)

FileContent тип *base64Binary*

error тип *string*

putFile

Описание: Запазва файл на сървъра. Файлът се запазва в директория, дефинирана като коренова директория за файлове с данни и дефиниции от WSDL сървъра на системата

Тип на операцията: *Request-response.* Крайната точка получава съобщение и изпраща съответно съобщение.

Вход: putFileRequest (soap:body, use = literal)

FileName тип *string*

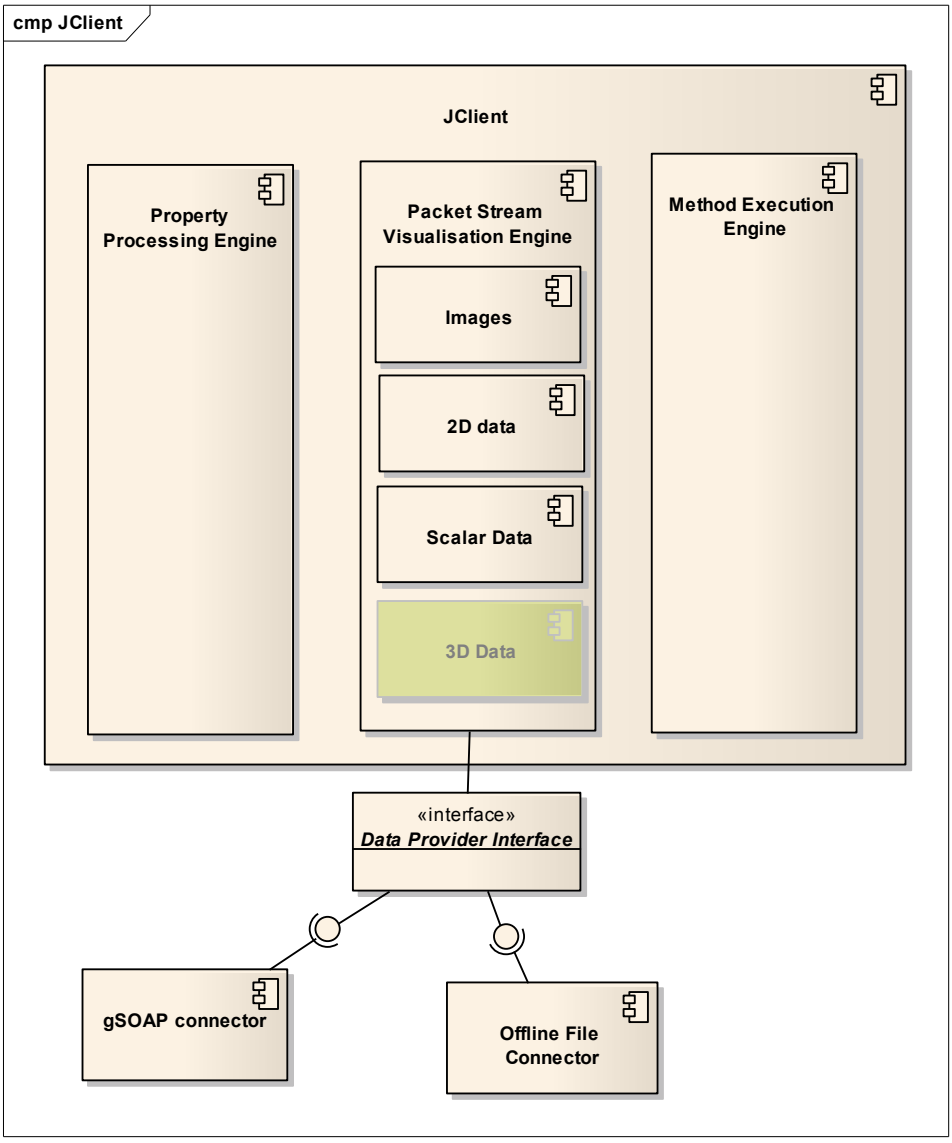
FileContent тип *base64Binary*

Изход: putFileResponse (soap:body, use = literal)

error тип *string*

Глава 3 – Архитектура на клиента

Клиентът може да бъде разделен на три основни модула: модул за визуализиране и промяна на свойствата; визуализатор на стрийм данните; модул за изпълнение на команди.



Диаграма 2. Архитектура на клиента

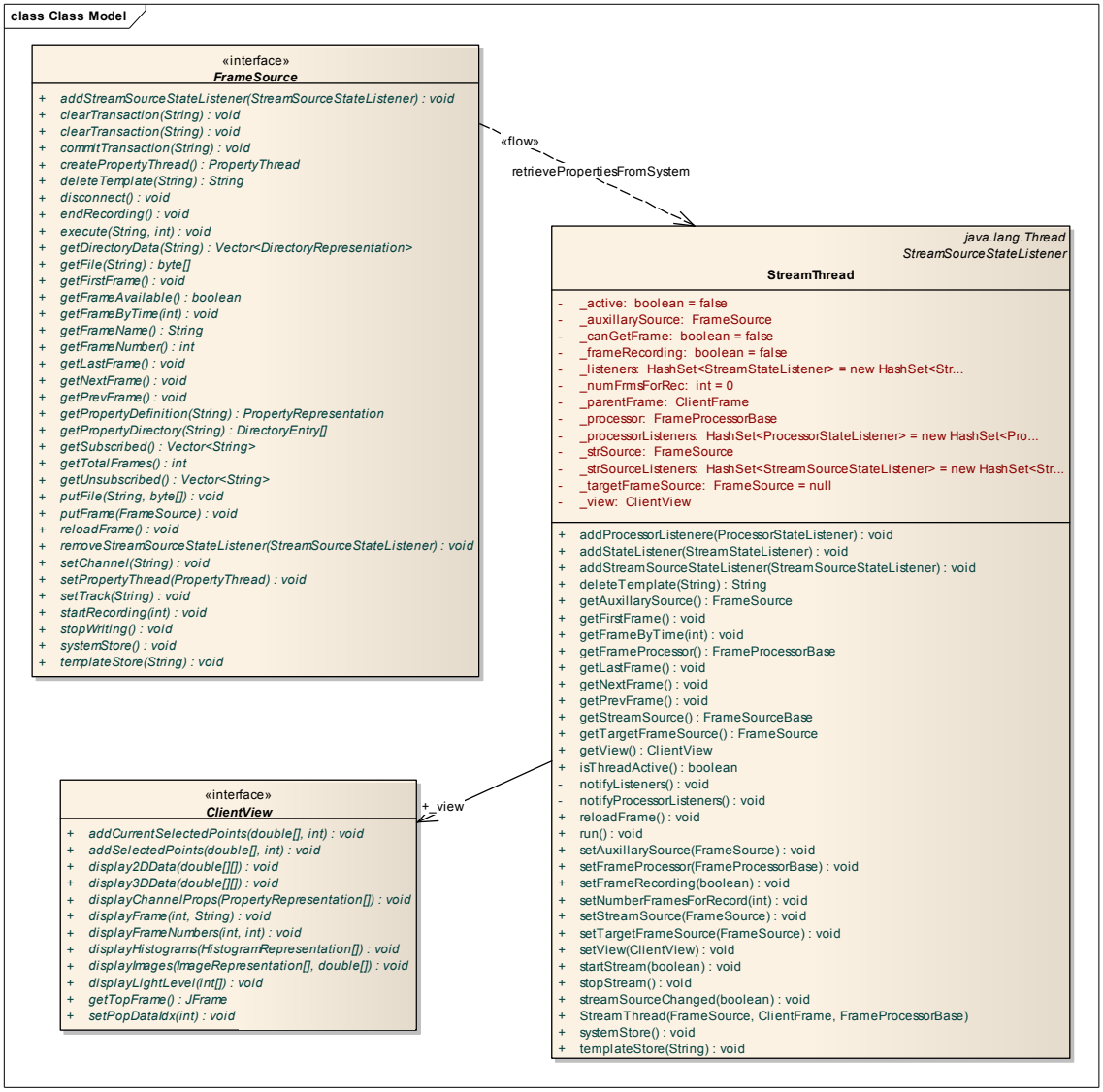
3.1.Property processing engine

Това е модулът, който се грижи за показването на конкретните свойства на системата. Той извлича наличната йерархия от свойства от съответното устройство. Клиентът не знае предварително каква е структурата на системата и какви свойства има в нея. Визуализация на системните характеристики на устройството става на базата на извлечените, от конкретната система, йерархична структура и описанието на нейните свойства. Модулът извлича още и дефиницията на свойствата. Дефиницията на свойствата дават по-конкретна представа на клиентското приложение и на потребителя за системата, с която работи. Модулът за визуализиране и промяна на свойствата позволява още и редактирането на свойствата, съответно на типа им. Типа на свойствата не е предварително известен, а се извлича от системата чрез дефиницията на свойствата. Показването на свойствата на системата в клиентското приложение става в йерархична структура, съответна на йерархичната структура извлечена от конкретната системата. Нивата от йерархията се визуализират като нива с табове в клиента. Всеки таб съдържа информацията (свойствата), съответстваща на получените данни от системата за конкретното ниво от йерархията (Диаграма 3).

Диаграма 3.

3.2. Packet Stream Visualisation Engine

Това е модулът, който се грижи за визуализирането на данните, получени от стриймовете. Клиентът има конкретно място за визуализирането на различните по тип данни идващи от пакетите със синхронни данни. При наличието на конкретни данни от пакет клиентът ги визуализира в съответната секция в графичния интерфейс спрямо типа на данните. Клиентът не се интересува предварително от типа на данните, които системата обработва. Той има определени места за различни типове данни. При извличането на данните от идващите пакети, модулът се грижи за мястото им на визуализиране спрямо типа им. Не е задължително в пакетите да има всички типове данни. Идеята на модула е да визуализира данните, които са налични като не се интересува предварително от техния вид и каква точно информация носят (Диаграма 4)



Диаграма 4. Визуализиране на данните идващи от пакети от системата

Клиентското приложение визуализира следните видове данни идващи от стриймове:

3.2.1. Images

Ако в пакетите има данни за снимки, клиентското приложение ги визуализира на определеното за това място в графичния потребителски интерфейс. Клиентът може да визуализира една или повече снимки в зависимост от това колко на брой снимки идват от пакетите на конкретната система. Броят на снимките не е дефиниран предварително. Той се определя по време на изпълнение и зависи от конкретната система. Ако снимките са повече от една в пакета с данните те се визуализират като отделни табове в секцията за снимките. Максималния брой на снимките в таб може да е повече от един. Това зависи от конкретните настройки на клиентското приложение.

3.2.2. 2D Data

Ако в пакетите на конкретната система има графични двумерни данни, те се визуализират в специална секция от графичния потребителски интерфейс, в която се визуализират всички данни от този тип. Тази секция представлява координатна система с две оси (по x и по y), в която се разполагат двумерните данни идващи от пакетите на системата (ако има такива). Броят на двумерните данни не е известен предварително. Той се определя по време на изпълнение и зависи от информацията идваща от пакетите. Ако има повече от един на брой такъв тип данни в пакетите те се визуализират с различни цветове, което позволява тяхното разграничаване един от друг.

3.2.3. Scalar Data

Скаларните данни или данните от прости типове се показват в таблица на отделно място в графични потребителски интерфейс. Предварителния брой на скаларните данни не е известен. Големината на таблицата се определя по време на изпълнение в зависимост от скаларните данни идващи от стрийма на системата, с която клиентът работи.

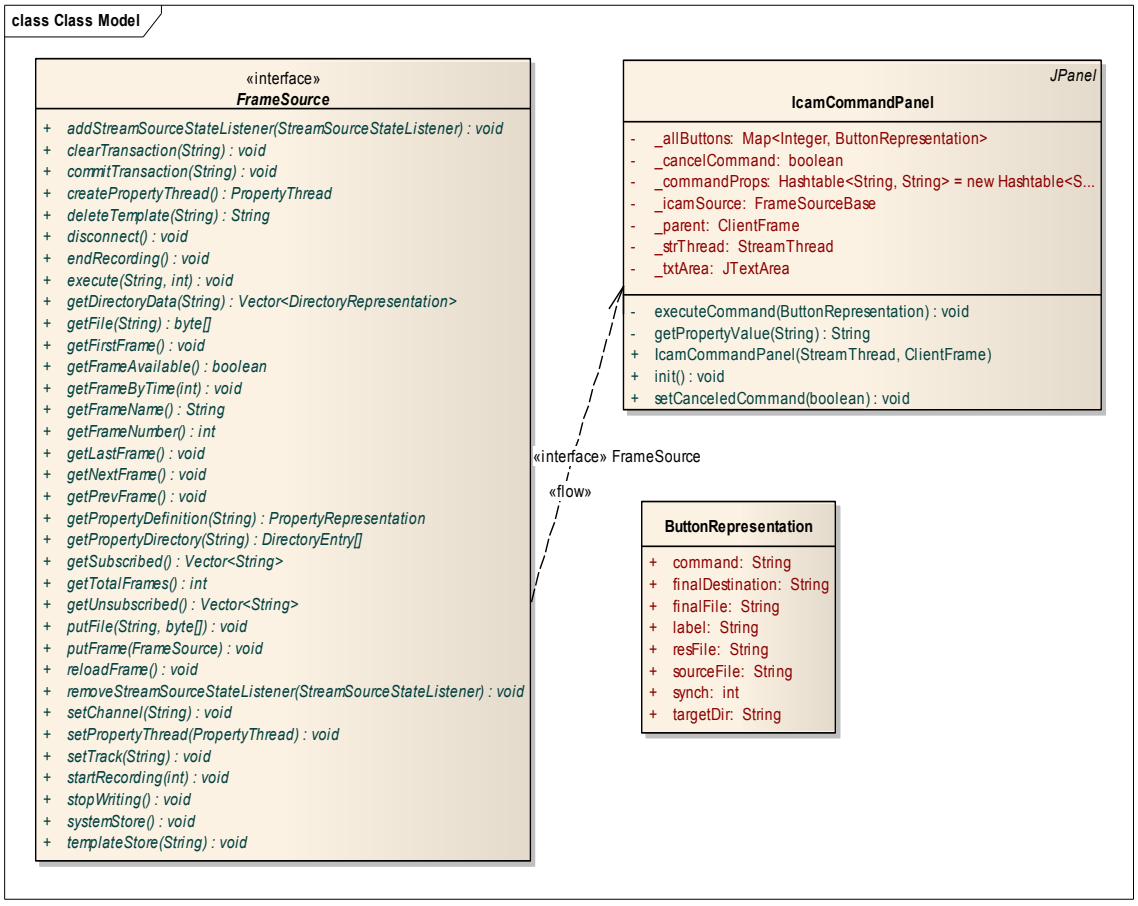
3.2.4. 3D Data

Визуализацията на тримерни данни също се извършва на отделно място в графичния интерфейс на приложението. Ако такива данни идват от пакетите на системата, то те се показват на това предназначено за тях място. Тримерните данни се

визуализират с помощта на координатна система с три оси (x, y и z). Броят на тримерните данни не се знае предварително. Той зависи от данните в идващите пакети на конкретната система. Ако техният брой е повече от един те се визуализират с различни цветове, което спомага за лесното им разграничаване един от друг.

3.3. Method Execution Engine

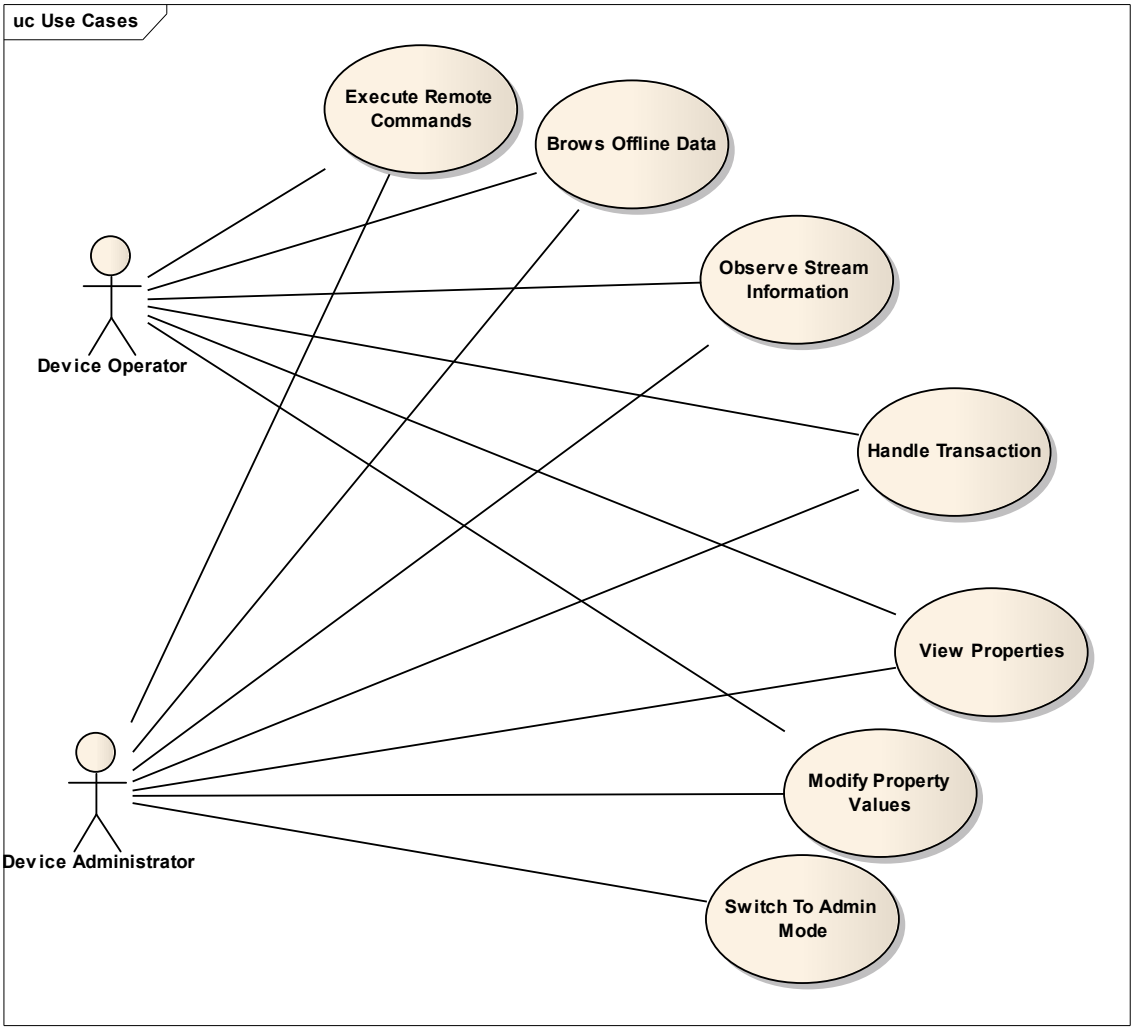
Този модул се грижи за изпълнението на различни команди на системата през клиентското приложение. Това става с помощта на бутони в клиента. Всеки бутон има описание, което носи информация за конкретната команда към системата. Описанието на бутоните се намира в конфигурационен файл. Чрез този модул се осъществява управлението на работата на системата отдалечено. Пример за такъв бутон може да бъде бутон, който при натискане рестартира системата чрез изпълнението на командата reboot.



Диаграма 5. Изпълнение на команди на системата

3.4.Use Cases

Основните потребителски роли са две – оператор и администратор на системата от вградени устройства.



Диаграма 6. Use Case диаграма

3.4.1. Оператор на устройството

Операторът изпълнява основните функции със устройството. Той може да вижда йерархията и организацията на системата и нейните свойства, да променя тези от тях, които не са разрешени само за адинистратор и не са само за четене, да извлича потоковите данни, да изпълнява команди към системата, да работи в offline режим и да работи с транзакции.

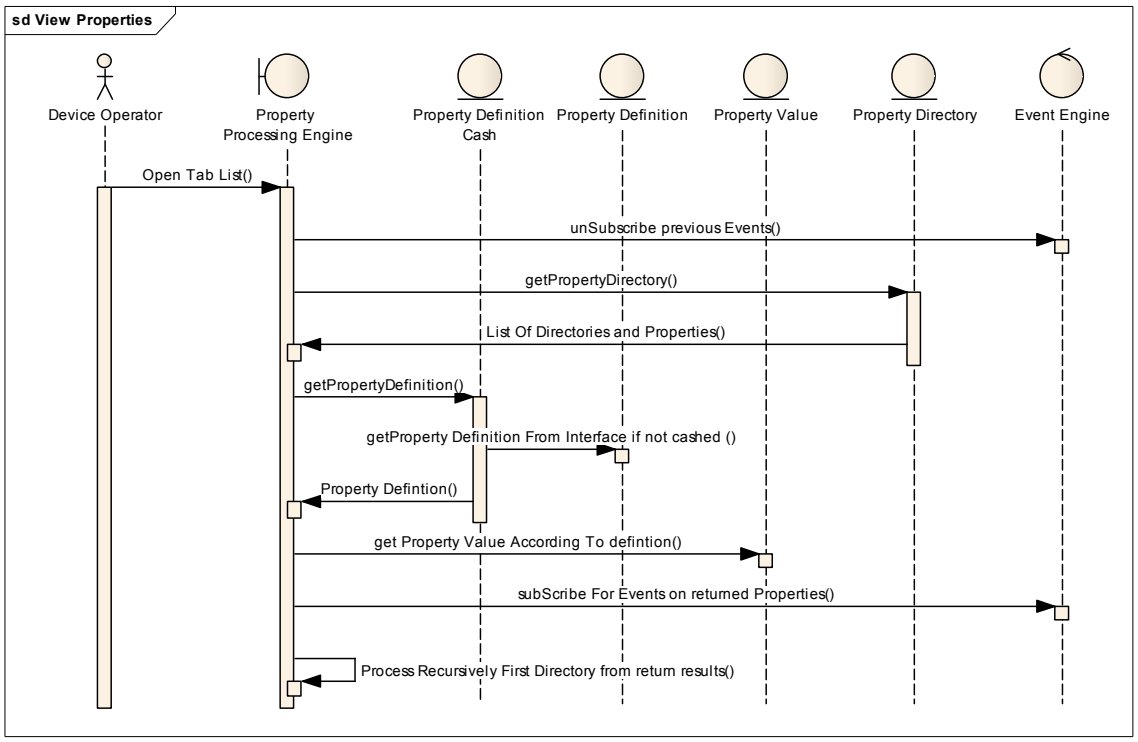
3.4.2. Администратор на увтройството

Администраторът може да изпълнява всички функции на оператора на системата. Освен тях администратора може още да преминава в администраторски режим и да променя свойства, които могат да бъдят променяни само в администраторски режим на работа.

3.5. Основни функции на клиентското приложение

3.5.1. Извличане на свойствата на системата

Тази функция се извършва при всяко преминаване през ниво от йерархичната структура на устройството. Клиентското приложение извиква от системата всички свойства, които се намират на това ниво. Извлича и дефиницията им. На базата на дефиницията на всяка свойство се извлича и неговата стойност. Информацията за свойствата се извлича рекурсивно за всяко подниво на текущото от системата (диаграма 7).

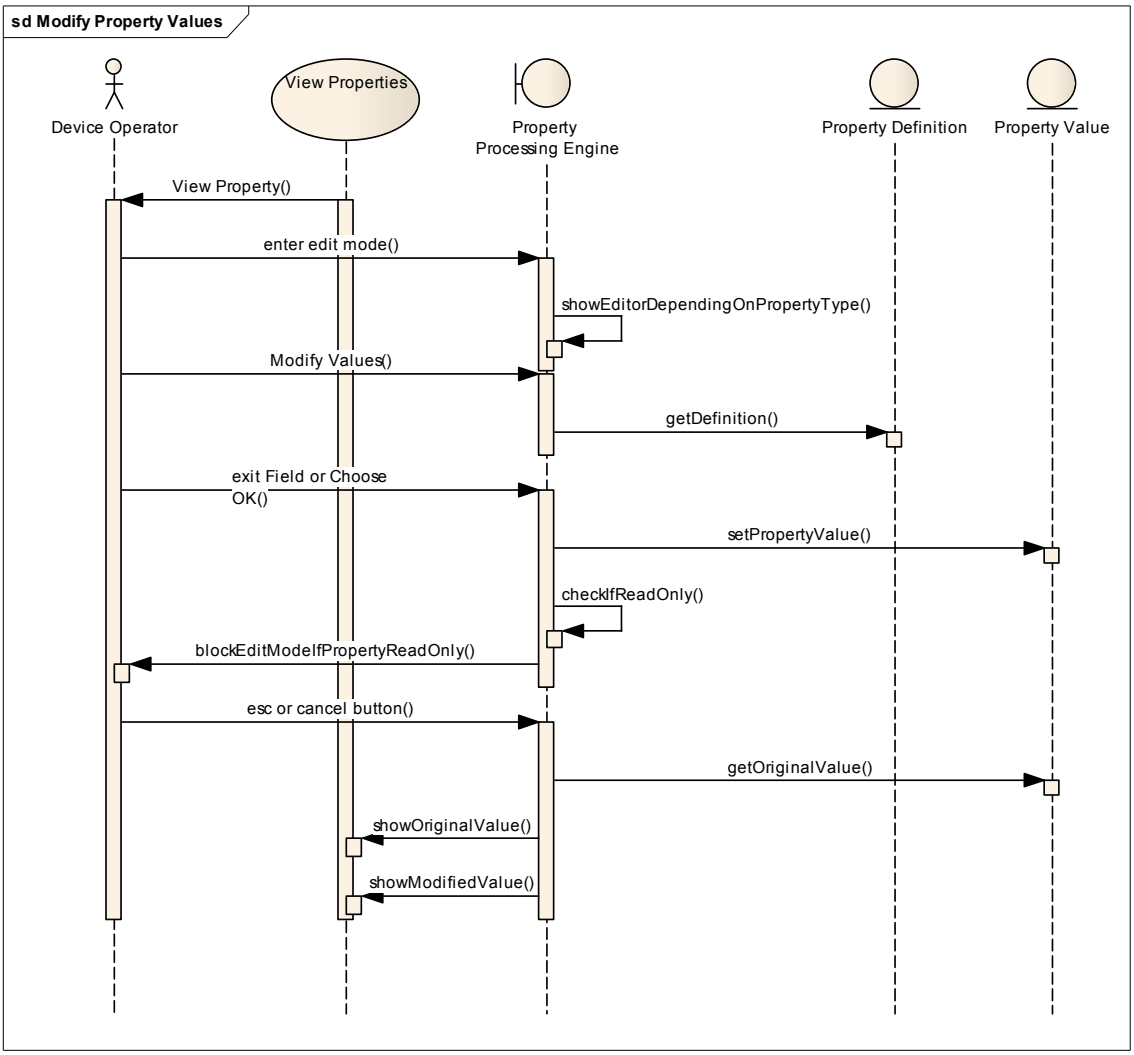


Диаграма 7. Извличане на свойства

3.5.2. Промяна на стойност на свойство

Когато потребителят иска да промени стойността на дадено свойство основното, което се прави е извличане на дефиницията на свойството. Ако то е разрешено само за четене

се блокира промяната на стойност. В противен случай се променя свойството в графичния интерфей и в системата и се показва новата стойност на свойто (диаграма 8).

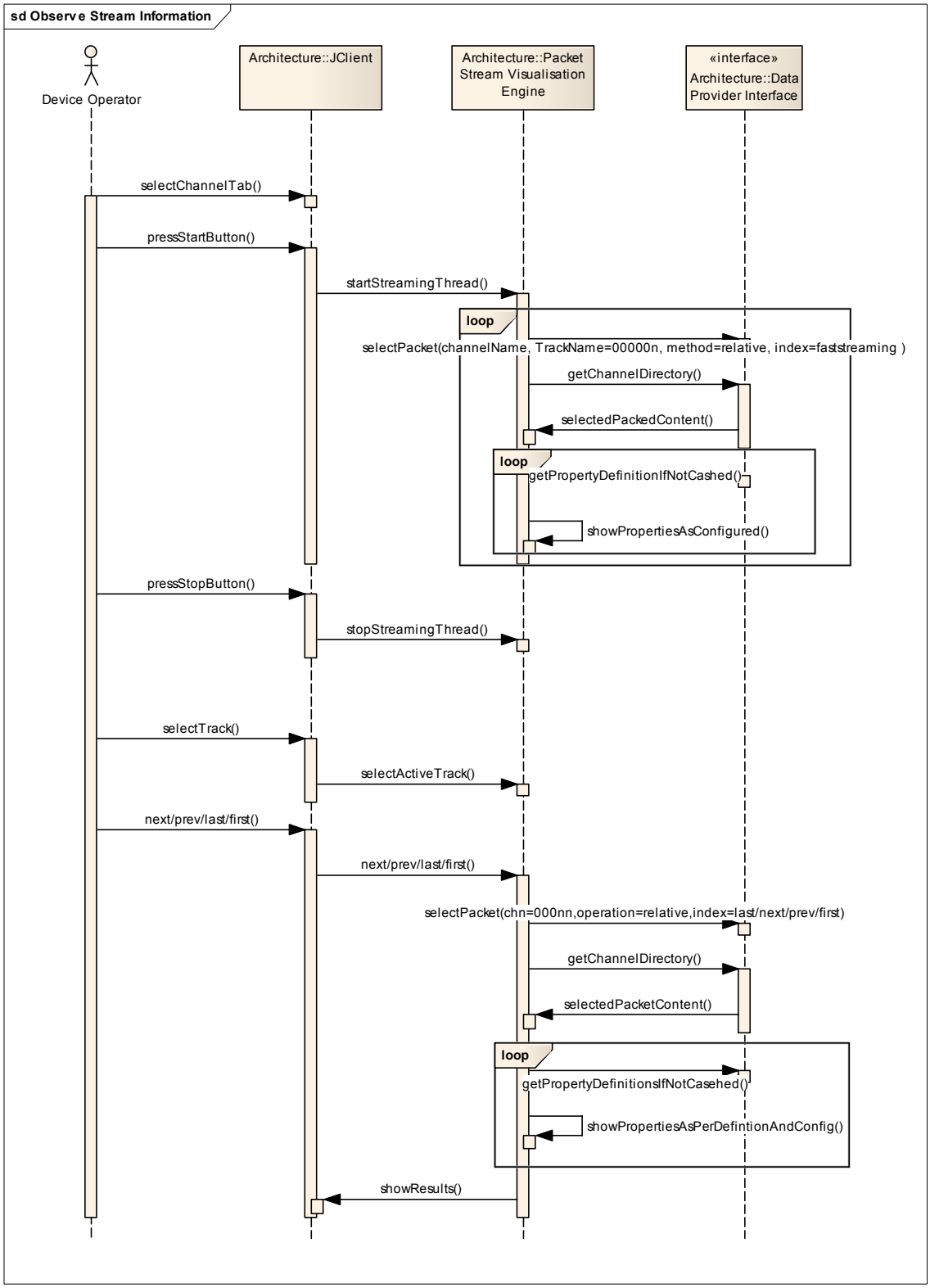


Диаграма 8. Промяна на стойност на свойство

3.5.3. Работа с потоковите данни

Работата с потоковите данни основно се извършва по два начина. Чрез стартиране на постоянен потоко за извличане на данни или чрез извличане на конкретни данни от системата. Стартирането на постяния поток става чрез „Start" бутон. При натискането на бутона се изличат последователно пакети с данни от системата; дефинициите на свойствата в пакета с данни и техните стойности. Процедурата спира при натискане на бутон „Stop". При натискане на бутон за извличане на конкретни данни от системата се

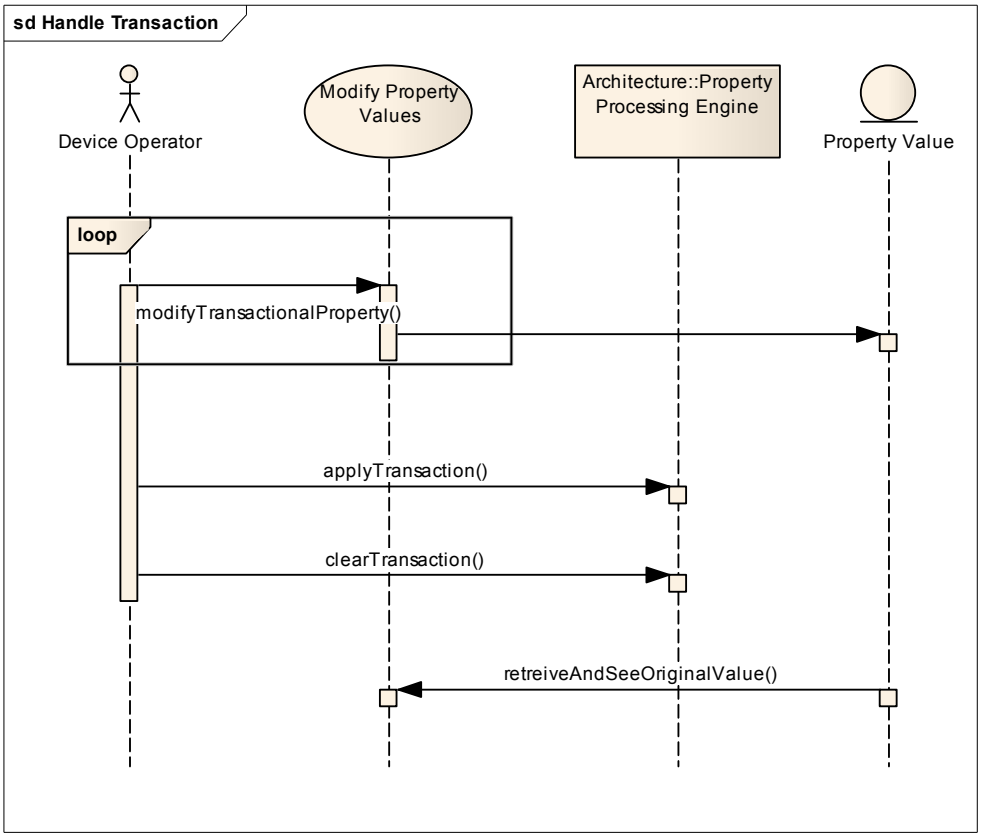
извършват еднократно същите действия за пакета, отговарящ на поисканиет данни (първи пакет, последен пакет, пакет извикан по време и т.н.) (Диаграма 9).



Диаграма 9. Извличане на потокови данни

3.5.4. Работа с транзакции

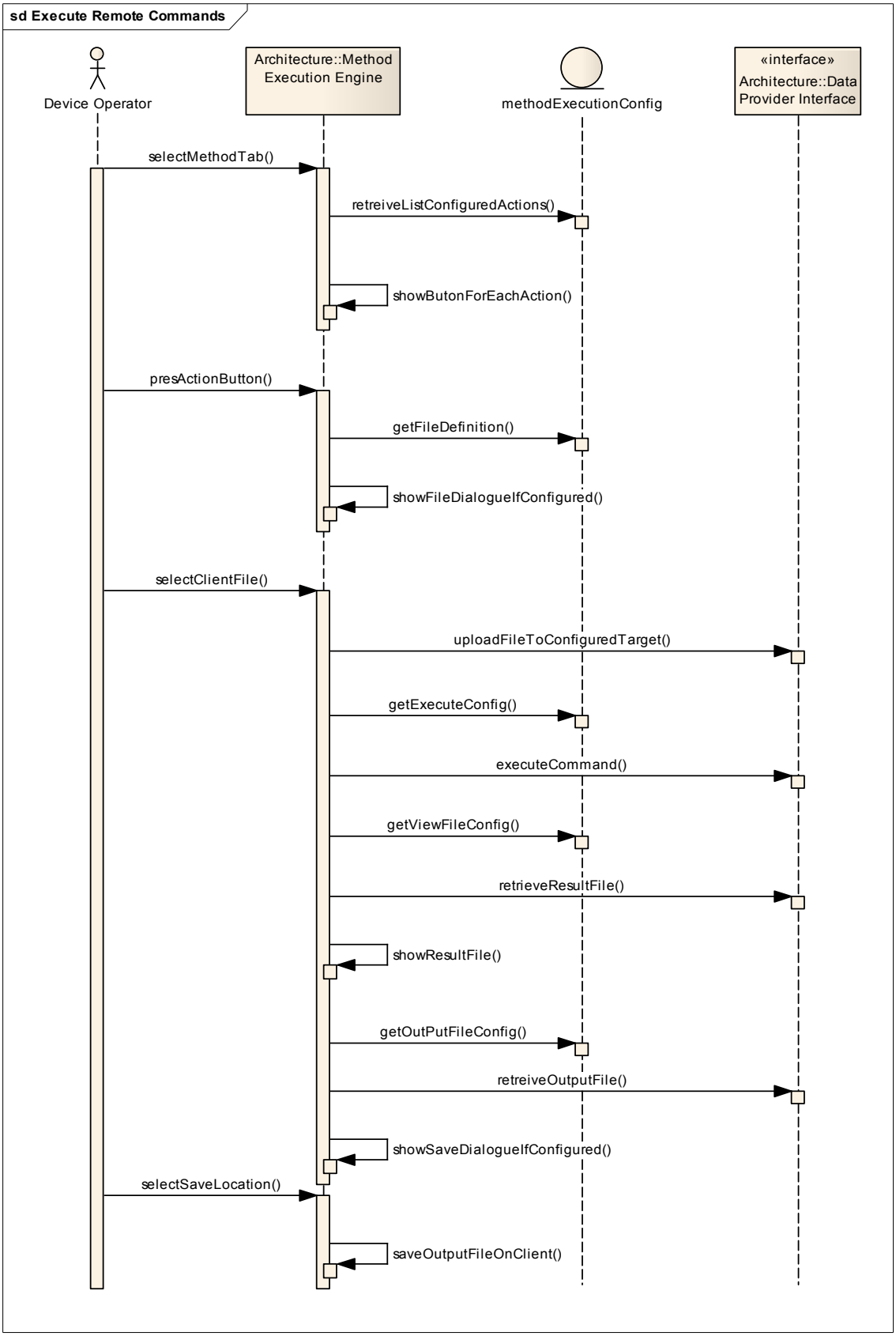
При промяна на транзакционно свойство, новата стойност може да бъде запазена трайно в системата или да бъде върната предишната стойност на свойството. След извършване на едно от двете действия се излича запазената в системата стойност и се показва в интерфейса на приложението (Диаграма 10).



Диаграма 10. Работа с транзакции

3.5.5. Изпълнение на команди към системата през клиентския интерфейс

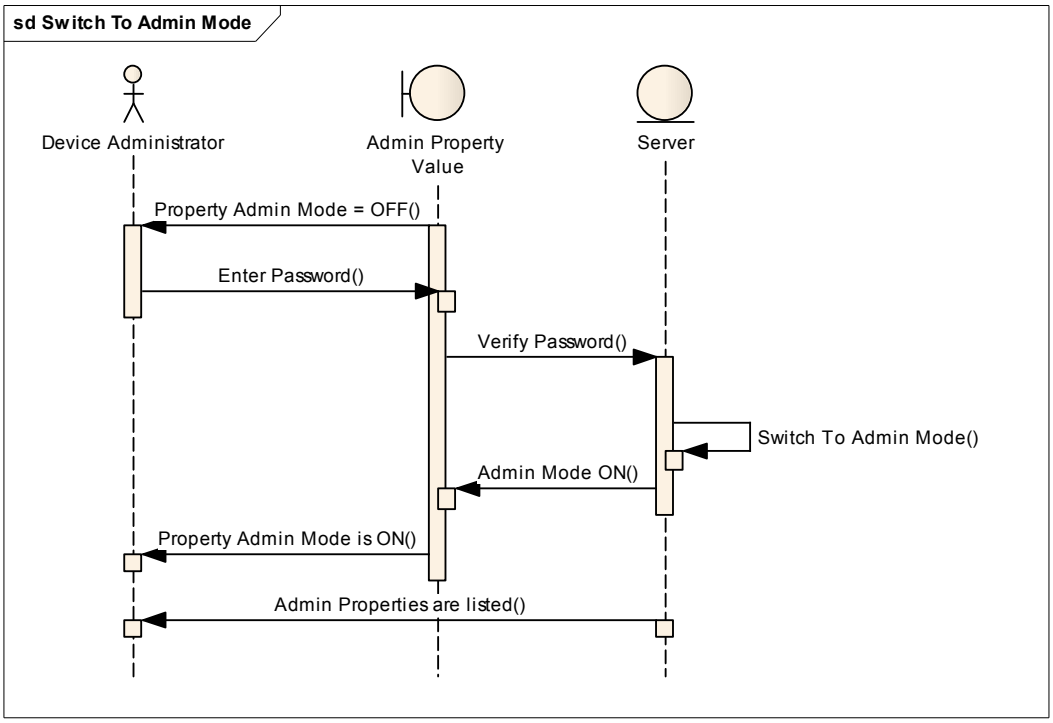
Тази функция се осъществява чрез команди бутони, разположени в интерфейса на клиента. Най-напред клиентското приложени извлича характеристиките за бутоните като име, команда, която ще изпълнява и т.н. от конфигурационен файл. След това бутоните се визуализират в интерфейса на приложението. При натискане на даден бутон се извършват следните действия: взимане на файл, описан за съответния бутон от конфигурационния файл или чрез показване на диалог за избиране на файл; изпращане на този файл към системата; изпълнение на командата, описана в конфигурационния файл за този бутон; ивзличане на резултат от изпълнението (ако има такъв); показване на резултата от изпълнението; извличане на изходен файл; запазване на изходния файл на системата. (Диаграма 11).



Диаграма 11. Изпълнение на команди към системата

3.5.6. Преминаване в администраторски режим на работа

Преминаването в администраторски режим се осъществява чрез промяна на свойство, което показва режима на работа. За да бъде преминаването успешно се въвежда парола като стойност на това свойство. Ако паролата е валидна, преминаването е успешно и се установява администраторски режим на работа (Диаграма 12).



Диаграма 12. Преминаване към администраторски режима на работа

Глава 4 – Структура на клиента

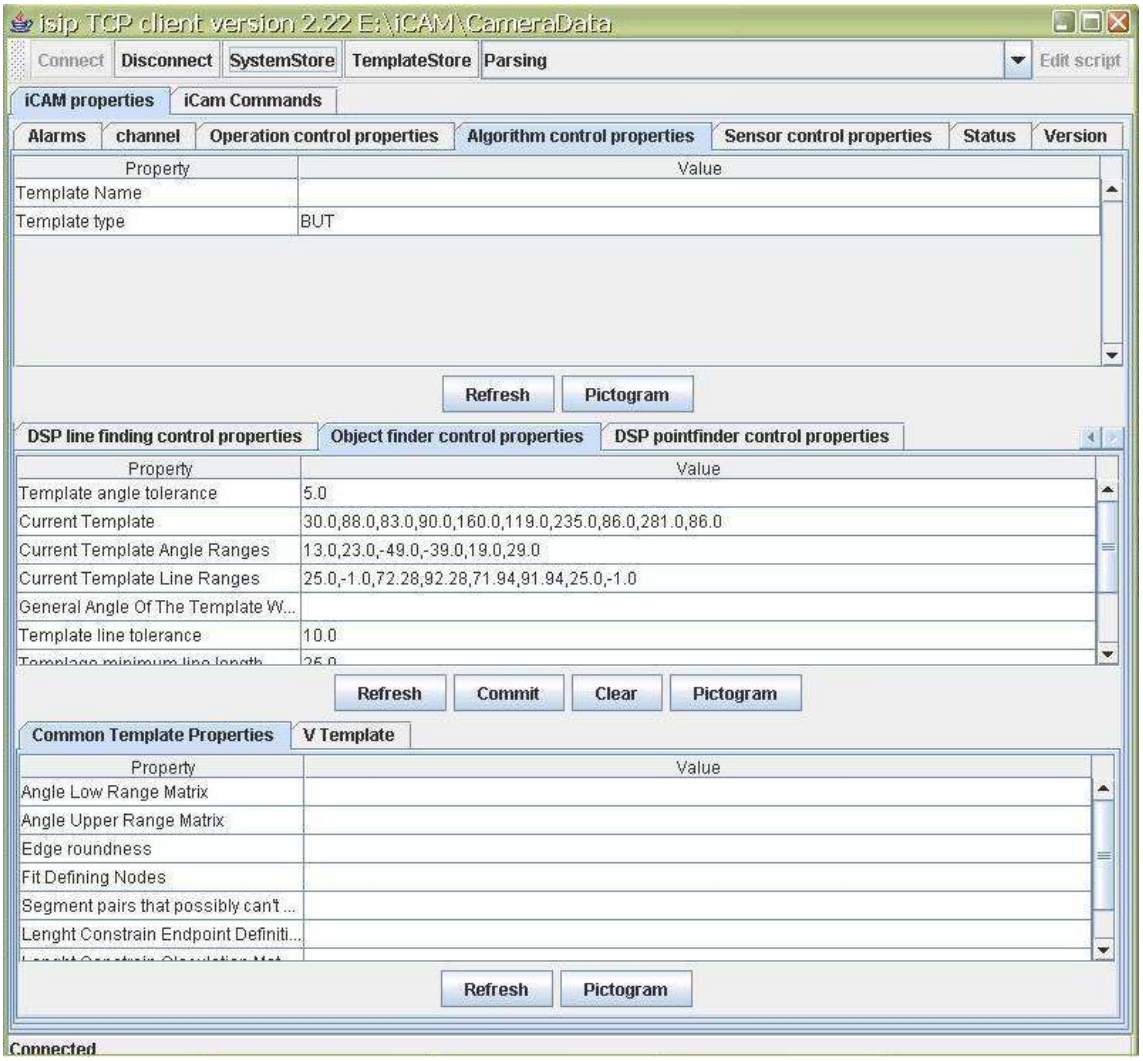
4.1. Важни характеристики на клиентското приложение

Основната цел на универсалния клиент, разглеждан в тази дипломна работа е да следи работата и състоянието на различни по вид устройства, така че да се постигне лесно и удобно администриране и управление на различни видове системи. Клиентът представя данните на даденото устройство чрез така наречените свойства, които описват конфигурацията на конкретната система.

Свойствата представляват информация за самата система - какво е нейното текущото състояние (например температура, интензитет, свободна памет и др.), какви са резултатите от определена операция (намиране на профил, брой пакети с данни, успешно или не преминаване в администраторски режим на работа и др.).

Следенето на устройството се осъществява на базата на комуникационния протокол, описан по-горе, чрез извличане на свойствата, а конфигурирането и контрола на устройствата – чрез промяна на определени свойства през интерфейса на клиентското приложение.

Визуализацията на структурата на системата се извършва след извличане на йерархията на конкретното устройство, с което клиентът работи. Йерархичната структура на системата в клиента се представя чрез йерархия от табове. Нивото на таб в графичния интерфейс на клиента отговаря на нивото на съответните данни в системата. Информацията за свойствата от системата (с изключение на свойствата идващи от стриймовете) се разполага в таблици под съответния таб от йерархията (фиг.1). Всичко, което се визуализира в клиента се определя от конкретната система с която работи клиентското приложение. Дори и имената, с които свойствата се визуализират се определят от системата. Клиентът не знае предварително нищо за системата и нейните данни. Той визуализира това, което идва като информация, извлечена от системата чрез комуникационния протокол. Единственото, което клиентът знае е начина по който да визуализира идващата информация. Това фактически представлява графичното изображение на системата. Например клиентът визуализира свойствата от системата в таблица с две колони – за името и за стойността на свойството. Конкретните стойностите за тези две колони както и броя на редовете на таблицата зависят от системата.



Фиг. 1 Примерна йерархична структура на система в графичния интерфейс на клиента

Този начин на организация на данните позволява на клиентското приложение да не се интересува от конкретните свойства и характеристики на системата, чийто контрол ще се осъществява през него. Клиентът се интересува единствено от информацията, която му е необходима за визуализирането на данните т.е. от информацията, която се извлича от йерархичната структура и дефиницията на свойствата на системата. Например в повечето системи има свойства, които не е разрешено да се променят и са само за четене (така наречените „read only" свойства) и такива, които е разрешено да се променят. Ако тази информация е описана в дефиницията на свойствата на конкретната система то свойствата, които са „read only" няма да могат да бъдат променяни през клиентеското приложение.

Един от важните елементи на дадено свойство е неговия тип. Клиентът трябва да знае с какъв тип данни работи, за да бъдат те визуализирани правилно. Така например float и int числа се визуализират като числа в таблицата на съответното свойство. Масивите (от float и int числа) се визуализират като поредица от числа разделени със запетая. Има свойства със специален тип като например enumeration (изброителен тип), динамични списъци и др. Такива свойства се визуализират като символни низове. Има и специален тип свойства като снимки, двумерни и тримерни свойства, които се визуализират на специални места в клиентското приложение. За тяхното визуализиране клиентското приложение използва и конфигурационен файл, специфичен единствено за клиента, който указва точното място и начин на визуализиране на тези специални свойства (като цвят, размер, прозорец, точки или линии, и т.н.). В това се изразява и универсалността на клиентското приложение. То не се интересува от това с каква система работи и какви устройства има тя, а само от нейните свойства и йерархия. Клиентът показва определена информация, която той знае как и къде трябва да разположи. Ако такава информация бъде извлечена от системата тя се визуализира. Например клиентът знае, че свойство от тип image е снимка, която трябва да бъде показана в даден прозорец. Ако при извличане на данни той получи свойство от този тип, то данните на това свойство се разполагат в съответния прозорец.

4.2. Работа със системата през клиента

Основното, което е важно за потребителя, работещ с приложение за достъп, контрол, визуализация и администриране на дадена система е средствата с които разполага за работа със системата, информацията, която получава и начина по който е организирана тя, средствата за манипулиране и настройване на системата, както и сигурен и защитен достъп до данните. Разглежданото клиентско приложение позволява тези изисквания да бъдат удовлетворени.

Потребителят получава информация за системата на базата на предоставените свойства. Всяко едно от тях описва конкретно състояние или характеристика за системата и показва стойност валидна за текущото състояние на системата. Йерархичната структура на табовете описана по-горе позволява потребителят лесно да се ориентира за структурата на цялостната система и за нейната организация. При избиране на определен таб от клиентеското приложение се извличат всички текущи състояние на свойствата, вътрешните табове и съответните им прилежащи свойства на текущо избрания таб. Това дава възможност на потребителя да следи за състоянието на

системата в реално време. Свойствата от дадена таб се извличат още и при натискане на бутон „Refresh”, разположен най-отдолу за съответния таб (под таблицата със свойствата – фиг.1). Това е удобен начин за потребителя да извлече необходима информация в даден момент без да се налага да преминава през таба отново.

При необходимост потребителят може да промени състоянието на дадено свойство чрез промяна на неговата стойност в таблицата. Клиентското приложение е проектирано така, че на базата на протокола за комуникация да уведоми системата за тази промяна и да извърши съответните действия.

Един основен въпрос при разработката на клиентски приложения е дали те осигуряват правилния достъп до данните и правилната манипулация с тях, така че да не се наруши целостта на системата или нейната работа. В повечето системи има данни, които не всеки потребител може да променя, или такива, които дават определена информация, но не могат да бъдат променяни. От голямо значение още е и типът на данните, които потребителят желае да промени. Присвояването на грешен тип данни може да доведе до сериозни проблеми или неточности в работата на системата и не трябва да бъде позволено. Клиентското приложение решава проблемите със сигурния достъп до данните чрез следните похвати:

- свойства, които са предоставени само за четене („read only”) и служат основно за визуализиране на важна информация не могат да бъдат променяни като достъпът до полето им в таблицата е забранен. Потребителят само може да вижда стойността му.
- всяко свойство има определен тип, описан в съответния xml файл. В този файл има описани още и така наречените: валидатор по време на писане и валидатор по време на присвояване. Първият проверява текущата стойност още докато потребителят я въвежда в полето от таблицата. Ако символа, който потребителят се опита да въведе е грешен според валидатора той не се разрешава и се игнорира. Валидаторът по време на присвояване проверява стойността, въведена вече в полето на свойството, при опит тя да бъде присвоена. Ако стойността не е допустима според валидатора не се разрешава тя да бъде присвоена на свойството. Ако тези валидатори не са описани в дефиницията на свойството и няма данни за тях, то валидността на въведена стойност за дадено свойство се проверя чрез неговия тип. Така например за свойства от числов тип int не е разрешено да се присвоява стойност, която

съдържа символ различен от число или стойност, която излиза от границите на int типа.

- повечето системи имат различни режими на работа като потребителски и администраторски. Ако дадено свойство е разрешено да се променя само в администраторски режим (това се описва отново в дефиницията му), то клиентът не разрешава то да бъде променено, ако не е в администраторски режим на работа. Режимът може да бъде променян и от самия клиент чрез правилно въвеждане на парола за съответното свойство указващо режима.
- други важни, но не задължителни характеристики на свойствата като зависимост от друго свойство, минимална стойност, максимална стойност и т.н. също играят важна роля при проверката за правилно присвояване на стойност. Ако те са описани в дефиницията на свойството приложението следи и за техните ограничения. Ако дадена стойност е по-малка или по-голяма от допустимите съответно минимална и максимална стойност за свойството, то тя се игнорира. Ако стойността е различна от стойност на свойството от което променяното свойство зависи, то тя се игнорира.
- специалните по тип свойства за клиентското приложение, като свойствата с изброим тип (enumeration) например, имат определено множество от допустими стойности. За да не бъде разрешено присвояването на друга стойност освен от това множество при опит за промяна на такъв тип свойства клиентът разрешава избор само на едно от разрешените стойности чрез падащ списък със допустимите стойности от множеството. Друго подобно ограничение на възможностите за присвояване на стойности на свойствата е при свойствата, които представляват масив. В таблицата със свойства техните стойности се показват последователно едно след друго, разделени със запетая. При промяна на такива свойства се показва диалогов прозорец с таблица с техните данните. Колоните на таблицата представляват размерността на масива на свойството, която може да бъде описана в дефиницията на свойството (ако липсва данни за нея се приема, че масива е едномерен). Потребителят има възможност да добавя или трие редове от тази таблица и да променя стойностите в нея. При натискане на бутон „OK” новите данни се присвояват на свойството и се показват отново като последователност, разделена със запетаи. Това не позволява на потребители да слагат невалидни символи в данните на масива или невалиден разделител между елементите му, което го

прави по-сигурен и по-надежден начин за промяна на такъв тип данни. Предимство още е и фактът, че потребителят може по-лесно да се ориентира коя точно стойност променя във масива, колкото голям или малък става той и т.н.

Благодарение на описаните методи за работа със системата през клиентското приложение, разработеният клиент напълно отговаря на изискванията и нуждите на потребителите в днешната динамична среда на сложните компютърни системи.

4.3. Реализация

С огледа на машинна независимост на приложението за неговата разработка е избрана Java като платформа. Езикът за програмиране Java е обектно ориентиран език, който има синтаксис подобен на C. Той е мощен, прост и удобен за създаване на чист и здрав код. Едно от основните предимства на Java, поради което е предпочитан за приложения като описаното в настоящата дипломна работа е неговата платформена независимост. Това означава, че написано веднъж, за Java платформа, едно Java приложение след това може да работи във всяка една система. Java платформата е интегрирана във всички основни операционни системи, в повечето web browser-и и дори в много електронни устройства като мобилните телефони. Друга основна черта на Java е сигурността. Платформата позволява на потребителите да използват дори ненадежден код, взет от интернет, стартиран в сигурна среда, където не може да причини никакви щети като например да инфектира системата с вирус, да записва файлове и т.н. Освен това всяко Java приложение може да бъде стартирано защитено чрез ограничаване на права, което предпазва от умишлено причинени вреди на системата. Това качество на Java платформата само по себе си я прави уникална и предпочитана. Още една важна характеристика на Java е това, че е мрежово ориентирана. От гледна точка на програмиста това дава изключително удобен и лесен начин за работа със ресурси през мрежа и за създаване на приложения от типа клиент/сървър. Езикът Java е динамичен и разширяем. Кодът е организиран в обектно-ориентирани единици наречени класове. Класовете се съхраняват в отделни файлове и се зареждат в Java интерпретатора само когато са необходими. Това означава, че приложението може да прецени по време на изпълнение от кои класове има нужда и да ги зареди когато има нужда от тях. Това означава още, че програмата може динамично да се разшири като зарежда класовете, които са и необходими и по този начин разшири функционалността си. [6]

Подходящи средства за графични приложения са Java AWT (Abstract Window Toolkit) и Java Swing инструментите. Библиотеката AWT предоставя основните средства за създаване на графични потребителски интерфейси и за визуализиране на графики. Тя е основата, върху която може да се построи по-сложна функционалност. Библиотеката Swing е разширение на AWT. Тя предоставя възможността да се създаде графичен интерфейс, които да изглежда по един и същи начин на различните платформи и изгледа да бъде системно независим. Това позволява създаването на графични приложения, които да имат свой изглед, който да не се променя и да не зависи от конкретния изглед на системата, на която работи. С оглед на това Swing е преносима библиотека, която позволява много по-лесно създаване на графични приложения, които се държат по един и същи начин на всички платформи. Тази библиотека е много по-голяма и всеобхватна. Това я прави и подходяща за универсалния клиент, разглеждан в тази дипломна работа. [7]

Реализацията на клиента може да бъде разделена основно на три части, които са тясно свързани помежду си. Графичен интерфейс, комуникация със системата и работа с данните, визуализация и организация на получената информация.

Комуникацията със системата от вградени устройства се извършва на базата на комуникационния протокол описан във втора глава. За целта се създава комуникационен порт към адреса на системата, който отговаря на изискванията на комуникационния протокол базиран на AXIS. Цялата комуникация между клиента и системата се извършва през този порт чрез методите на протокола. Това става с помощта на FrameSource интерфейса, който създава връзката между графичния интерфейс на клиента и методите на комуникационния протокол. Той се грижи за преноса на данните от графичния интерфейс към протокола за комуникация. Работата с данните, тяхното визуализиране, настройването на системата през клиента се извършва чрез него и чрез основната нишка на приложението за извличане на данни - StreamThread. Тя се грижи за визуализирането и показването на потока от данните, идващ от протокола за комуникация. При стартиране на процедурата за извличане на данни главната нишка на приложението се обръща към методите на текущия наследник на FrameSource интерфейса за осъществяване на комуникацията със системата. Текущия наследник на FrameSource интерфейса извлича данните от системата, използвайки методите на комуникационния протокол и ги връща на основната нишка за данните. Получените данни нишката подава на друг важен интерфейс в системата – този който се грижи за тяхното визуализиране в графичния интерфейс. Това са

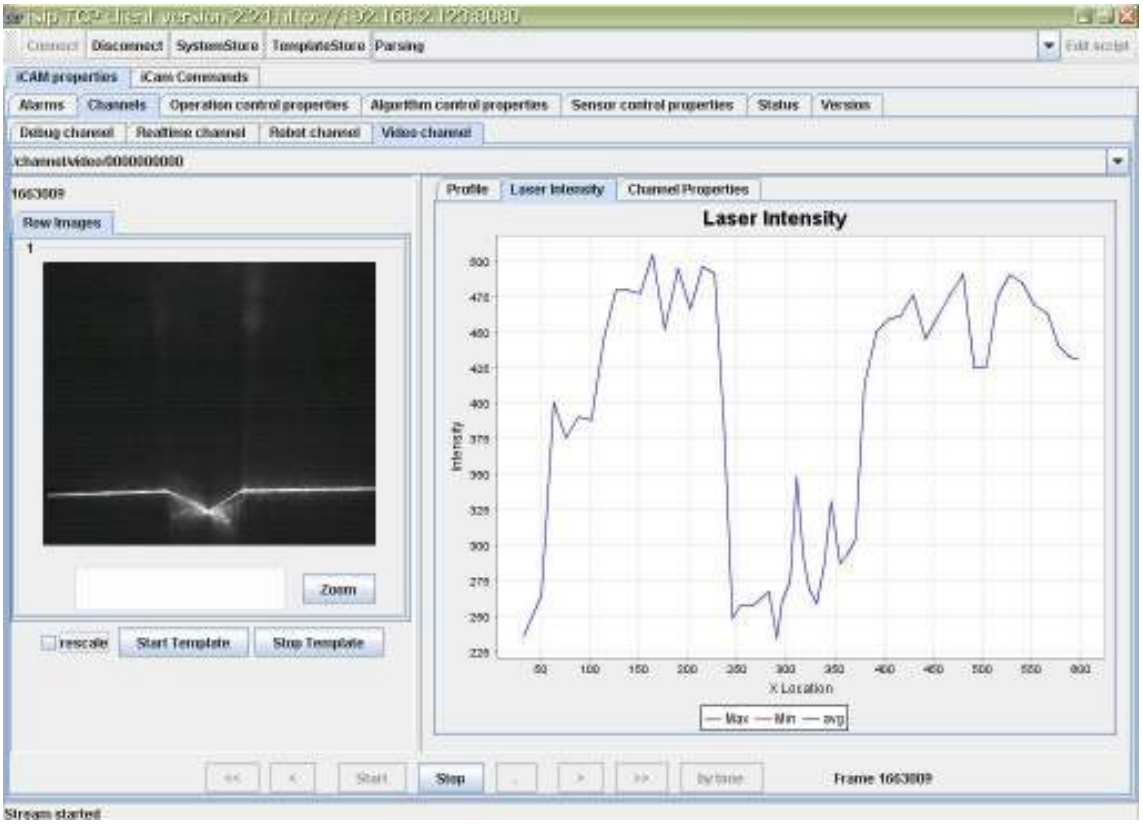
основните процеси в клиентското приложение, осигуряващи цялостен поглед върху системата.

4.3.1. Графичен интерфейс

Графичният интерфейс е начина по който потребителят вижда данните на системата. Основните компоненти в него са табовете и таблицата със свойствата и техните стойности. Таблицата към даден таб съдържа свойствата, които са в съответното място в йерархията на свойствата. Както беше описано по-горе свойствата имат различни типове и според това могат да бъдат разделени на два вида свойства – обикновени и специални. Обикновените свойства се визуализират в таблиците и могат да бъдат извлечени от системата чрез бутон „Refresh” разположен под таблицата или при преминаване през таб. Тук няма нужда от постоянен поток на данни, защото потребителят може да поиска извличането им чрез бутона. Поради тази причина комуникацията със системата не минава през основната нишка за извличане на данни на приложението. Когато „Refresh” бутона бъде натиснат текущия наследник на FrameSource интерфейса се уведомява и извършва извличането на данните. Това става чрез ActionListener класа от Java awt библиотеката, който следи за събитие възникващо при извършване на определени действия (в случая натискане на бутон). Когато получи съобщение за получаване на такова събитие се извършва извличането на данните за текущо избрания таб. Аналогично - при промяна на свойство от таблицата, тя получава известие и уведомява FrameSource наследника за извършената промяна. По същия начин се извършва и извличането на данните при преминаване на нов таб.

Интерес в повечето системи представляват специалните свойства. Обикновено те носят по-специфична информация за или от системата от вградени устройства. Универсалният клиента разглежда един набор от специфични свойства (като снимки, двумерни данни, тримерни данни – данните идващи от стриймовете), които визуализира на специални места в графичния интерфейс. Клиентското приложение показва някаква информация за тях само, ако бъдат извлечени такива данни от съответната система, с която клиентът работи. Клиентът не се интересува от това дали има такива данни в системата или няма. Той знае какво и кака да визуализира, къде да го разположи, с какви цветове, големина и т.н. и ако получи дадената информация я показва в графичния интерфейс. Именно обмяната на тези специфични данни със системата се извършва през основната нишка на приложението. За стартирането или спирането на потока от данни или за извличане на конкретни данни от системата

графичния интерфейс предоставя бутони за уведомяване на основната нишка за съответните действия. „Start” бутона стартира постоянен поток. „Stop” бутона спира потока от данните. „•” бутона извлича отново текущите данни. „>” бутона извлича следващите налични данни след текущите. „<” бутона излича данните преди текущите. „<<” бутона излича първите налични данни. „>>” бутона извлича последните налични данни. „by time” бутона извлича данните от определено време.

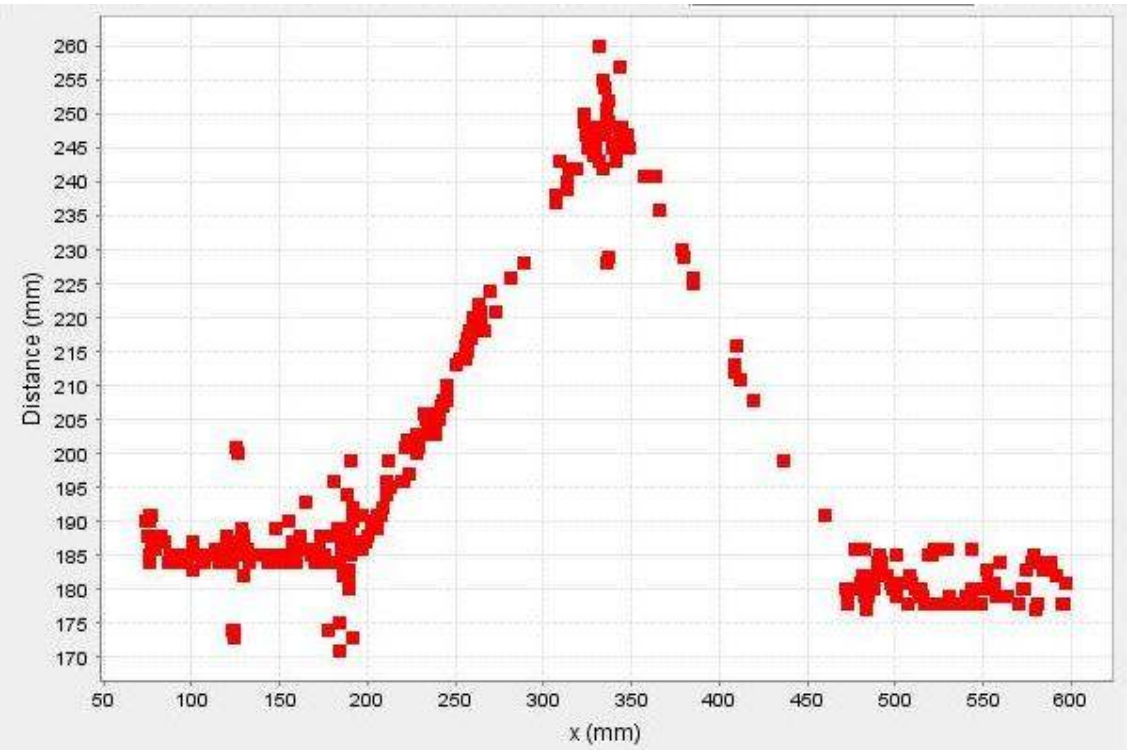


Фиг. 2 Визуализиране на данните идващи от потока за данни

Визуализирането на специалните свойства като двумерните и тримерните данни се извършва в диаграми с помощта на допълнителна Java библиотека – JFreeChart. JFreeChart е мощна Java библиотека за генериране на диаграми, таблици, схеми, хистограми, времеви измервания, термометри и много други. Поддържа прости графики, вертикално/хоризонтално разположени (може и с 3D ефекти), XY чертежи, комбинирани диаграми и много други. JFreeChart позволява изработване и на двойни чертежи, увеличаване на образа, обработка на събития от мишката, принтиране, както и записване на готовите графики в PNG/JPEG/SVG/PDF формат. Продуктът има версии за Windows и Linux. Тази библиотека предоставя лесен и удобен начин на разработчиците

да създават качествени диаграми в техните приложения. Множеството от възможности и свойства включва още гъвкав дизайн, който е лесен и удобен за разширяване както и възможност да се работи от двата вида приложения – сървърни или клиентски. JFreeChart поддържа още и много изходни типове, включително и от Java swing библиотеката като снимки, векторни графики и др. [8]

С помощта на JFreeChart двумерните данни се визуализират в координатна система. Това става като данните (x и y координатите за тези данни, идващи от системата през комуникационния протокол) се подават на класа XYSeries от библиотеката на JFreeChart. Този клас представя серия от нула или повече данните от вида (x, y). Тези серии се изобразяват на диаграмата (координатната система) чрез класа XYLineAndShapeRenderer, който се грижи за начина по който те ще бъдат показани - дали ще има линии между точките или не, дали ще се показват като определена фигура, с какъв цвят ще бъдат визуализирани данните и други. Тези характеристики са описани в конфигурационния файл на клиента. В него са описани всички серии от данни, които клиентът може да визуализира. Освен цвят и форма конфигурационният файл показва още дали данните ще бъдат визуализирани с някаква анотация, даваща определена информация за данните. Анотациите се показват чрез класа XYDrawableAnotation. При визуализирането на данните в координатната система е дадена възможността на потребителя да избира кои данни иска да се виждат и кои не. Това е постигнато чрез добавянето на „check box” за всяка серия с данни. Ако „check box” е избран, то данните се показват, в противен случай – не се показват. (фиг. 3)



Фиг. 3 Визуализиране на двумерни данни

Визуализирането на снимките се извършва чрез помощта на допълнителна библиотека за обработка на снимки – ImageJ. Данните за снимките се подават на метода за визуализиране като масив от byte числа. Библиотеката ImageJ е подходяща за обработката на данни от такъв тип, защото тя предоставя голям набор от методи за визуализиране, обработка, анализ, запазване, принтиране и т.н. на 8-битови, 16-битови и 32-битови снимки. С нейна помощ могат да се четат още и много файлови формати на снимки като TIFF, GIF, JPEF, BMP, DICOM, FITS както и сурови данни за снимки. Библиотеката може още да поддържа и серии от снимки, които се показват на един прозорец. Тази библиотека е мултинишкова, така че операциите, които консумират повече време като четене на снимка от файл могат да бъдат извършвани паралелно с други операции. Библиотеката дава възможност и за пресмятане на пиксели, мерене на разстояния и ъгли. Може да изчислява хистограма на снимките. Поддържа още и стандартните функция за работа със снимки като промяна на контраста на снимка, засилване или намаляне на цветовете, изчисление на медианите и т.н. ImageJ предоставя възможности и за геометрични трансформации на снимките като мащабиране, завъртане и преобръщане. Снимките могат още да бъдат приближени или отдалечени до 32 пъти. Библиотеката позволява още и работа в реални пространствени

координати. Тя дава възможност за работа с мерки в милиметри Библиотеката може да работи на всяка машина, на която има Java 1.4 или по-висока версия. Това я прави силна и подходяща за приложения от разглеждания тип. [9]

4.3.2. Комуникация със системата

Основната комуникация със системата се извършва както вече беше споменато чрез протокола за комуникация, чиито методи се извикват благодарение на FrameSource интерфейса, осигуряващ връзката между графичния интерфейс и системата от вградени устройства на базата на комуникационния порт. За целта FrameSource интерфейса предоставя набор от методи, към които клиентското приложение се обръща при нужда от информация от системата. Тези методи от своя страна използвайки порта за комуникация и методите на комуникационния протокол се обръщат към съответните методи на комуникационния протокол и получават от системата съответната информация. Създаването на порта за комуникация със отдалечената система става при натискане на бутона “Connect”. Тогава се създава порт към съответния адрес на системата чрез ICAMWebServiceLocator класа на комуникационния протокол След това се създава обект от клас, наследник на FrameSource интерфейса като в конструктора на този клас се подава създадения порт към системата. За визуализиране на табовете приложението се обръща към метода на FrameSource интерфейса getDirectoryData. В този метод се извършва извличането на йерархичната структура от системата чрез метода на протокола за комуникация getDirecotoryDefinition, извличат се още имената (или дефинициите) на компонентите в йерархията както и свойствата, които тя съдържа. Тези данните се преобразуват в подходящ вид за приложението и се връщат за визуализиране. Свойствата за визуализиране се намират чрез метода на FrameSource интерфейса getPropertyDefinition, който се обръща към съответния метод от протока за комуникация. Стойностите на свойствата се извличат чрез getProperty метод. Този метод използвайки java.lang.reflect библиотеката вика getProperty метод, съответен на типа на свойството (например getPropertyValueFloat, getPropertyIntArray и т.н.), който от своя страна се обръща към съответния метод на комуникационния протокол за извличане на стойността от системата. По аналогичен начин се извършва и присвояването на стойност на свойство в системата – чрез извиквания на подходящия метод спрямо неговия тип. Извличането на пакетите с данните от системата се извършва чрез методите getFirstFrame(), getPrevFrame, getNextFrame, getLastFrame(), reloadFrame, getFrameByTime() съответно за извличане на първи пакет, предишен пакет,

следващ пакет, последен пакет, текущ пакет или пакет по време. Тези методи се обръщат към метода на комуникационния протокол selectPacket, със съответния индекс за метода по който се извлича пакет и индекса на пакета. За първи пакет selectPacket се вика с метода на извличане „relative" и индекс на пакет -2, за предишен пакет индекса е -1, за следващ – 1, за последен – 2, за взимане на пакета по време метода е „timestamp", а индекса е число показващо времето на пакета, за извличане на текущия пакет се използва „frame" с реалния индекса на пакета. Това са основните методи, необходими на клиента за визуализирането и работата със системата. Всички допълнителни операции като извличане на файл от системата, пращане на команди към системата и др. са осъществени по подобен начин. При прекратяване на работата със системата клиентът извиква методът disconnect(). Той нулира порта към системата и по този начин прекъсва връзката с комуникационния канал към отдалечената система от вградени устройства

4.3.3. Визуализиране на данните

Визуализирането на данните се извършва чрез основната нишка за извличане на данни – StreamThread. Класа StreamThread е наследник на Java класа Thread от java.lang библиотеката. Методът run() е основният метод на нишката и в него протичат най-важните събития, които нишката изпълнява. При натискане на бутона „Start" през графичния интерфейс на приложението се стартира нишката, след което през определен период от време, синхронно в run() метода, се извличат данните от системата като се вика методът за извличане на следващите данни. Докато нишката е стартирана и има нови извлечени данни, то тези данни се подават от нишката на графичния интерфейс и се визуализират. При натискане на бутона „Stop" през графичния интерфейс се спира извличането на данните. В този случай нишката може да извлича данни чрез един от бутоните за извличане на текущия пакет от данни наново, за извличане на предишния пакет от данни, за извличане на следващия пакет от данни, за извличане на първия пакет от данни или за извличане на последния пакет от данни. Ако един от тези бутони бъде натиснат нишката извиква съответните данни чрез обръщение към съответния метод от текущия наследник на FrameSource интерфейса. Извлечените данни могат да бъдат различен брой всеки път и от различен тип. Нишката получава всички данни чрез различните методи на FrameSource интерфейса. Различните видове данни се визуализират и чрез различни методи от интерфейса за визуализиране на данни в графичния потребителски интерфейс - ClientView. Например снимки се визуализира

чрез метода displayImages, който приема списък от данни на снимки. Двумерните данни се визуализират чрез метода display2DData, който приема списък от данни за това, което се рисува на диаграмите. Данните за хистограмите се визуализират чрез метода displayHistograms, който приема списък от данни за хистограмите.

4.4. Бутони и команди към системата

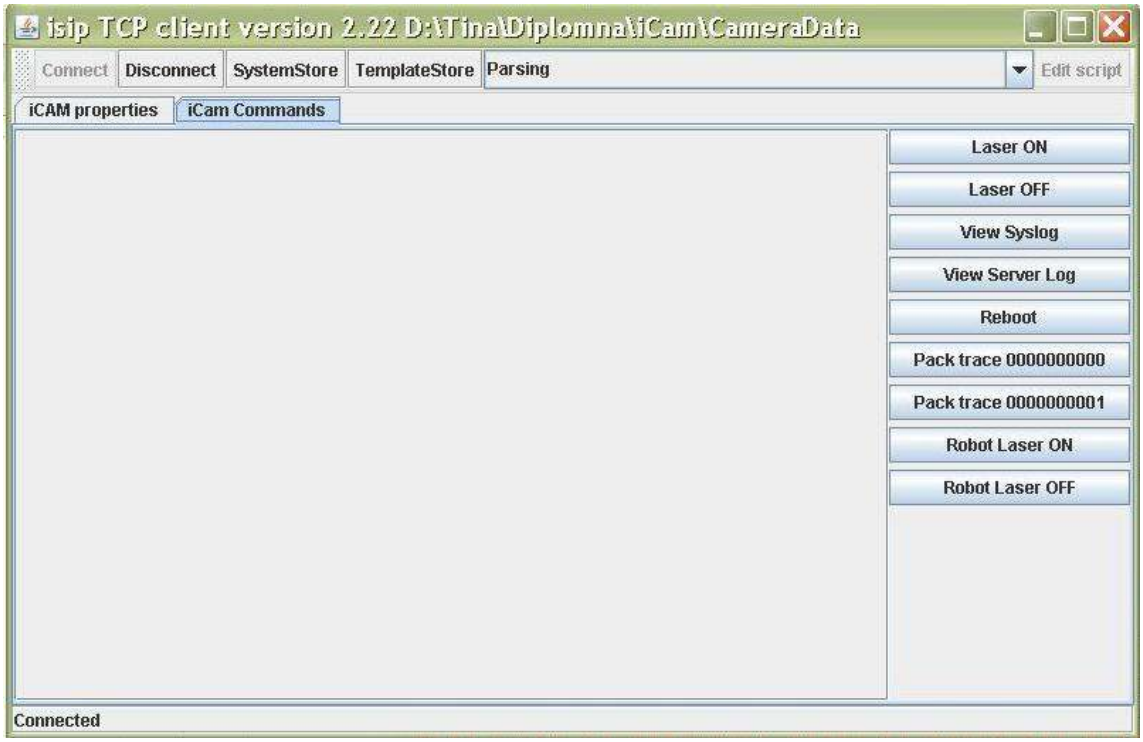
Клиентското приложения предоставя още един метод за управление и диагностика на системата от вградени устройства, с която работи – чрез бутони, изпълняващи определени команди на системата, използвайки протокола за комуникация. Описанието на бутоните и на командите, които те изпълняват се намира в конфигурационен файл - icam_command.properties.

icam_command.properties

```
button1.label=Laser ON
button1.command=/opt/bin/testsenderprocess SIMPLESTORAGE LASER_STATUS 1
button1.resultfile=
button1.synch=1
button1.sourcefile=
button1.targetDir=

button2.label=Laser OFF
button2.command=/opt/release/testsenderprocess SIMPLESTORAGE LASER_STATUS 0
button2.resultfile=
button2.synch=1
button2.sourcefile=
button2.targetDir=

button3.label=View Syslog
button3.command=
button3.resultfile=/var/log/icamlog
button3.synch=
button3.sourcefile=
button3.targetDir=

button4.label=View Server Log
button4.command=cat /var/log/icamlog | grep ICAM_SERVER: > /tmp/srvlog
button4.resultfile=/tmp/srvlog
button4.synch=1
button4.sourcefile=
button4.targetDir=

button5.label=Reboot
button5.command=reboot
button5.resultfile=
button5.synch=0
button5.sourcefile=
button5.targetDir=
```

Фиг. 4 Примерен properties файл за конфигурация на командни бутони

Този файл показва, че в клиентското приложени ще има 5 бутона за управление на системата чрез определени команди. Всеки бутон има определени характеристики като не всяка от тях е задължителна. Полето label е задължително за всеки бутон. То указва името с което бутонът ще бъде визуализиран в клиентското приложение. command представлява командата, която ще се изпълнява на системата. resultfile указва пътя до файл, който ще бъде извлечен от системата и показан в клиентското приложение. synch указва дали съответната команда ще бъде изпълнена в синхронен или асинхронен режим на изпълнение. sourcefile указва пътя до файл, който ще бъде изпратен на системата. targetDir указва пътят до директорията в системата където sourcefile-а ще бъде запазен. Ако в конфигурационният файл са описани всички полета най-напред се изпраща sourcefile-а до системата, след което се изпълнява командата и най-накрая се извлича резултата и се показва неговото съдържание в клиента. Това е удобен начин за приложението да изпълнява определени команди отдалечено и да управлява цялостната работа на системата. По този начин може например да се рестартира системата, да бъдат извлечени или изпратени необходими файлове и др. Изпълнението на командите отново се осъществява чрез методите на FrameSource интерфейса, който се обръща към съответните методи на комуникационния протокол. Бутоните на командите се разполагат в друг таб до root таба на системата. Бутоните са разположени от дясната страна на прозореца, а в средата е мястото, където се показва съдържанието на resultfile, ако има такъв извлечен файл.



Фиг. 5 Командни бутони в графичния интерфейс на клиента

Клиентското приложение използва рационално и максимално това, което му предоставя комуникационния протокол като данни и информация. Реално системата, която стои зад протокола не е от значение за клиента. Клиентът знае какво, как и къде да визуализира и да покаже. Ако системата дава информацията от която клиентът се нуждае, то тя се показва или обработва чрез интерфейса на комуникационния протокол. Ако не - се показва само това, което се предоставя от системата. Именно в това се изразява универсалната работа на клиенското приложение. Още един плюс за клиента в тази насока е и възможността, която той предоставя за работа в offline режим (виж приложение 1).

Глава 5 – Тестване и работа с клиента

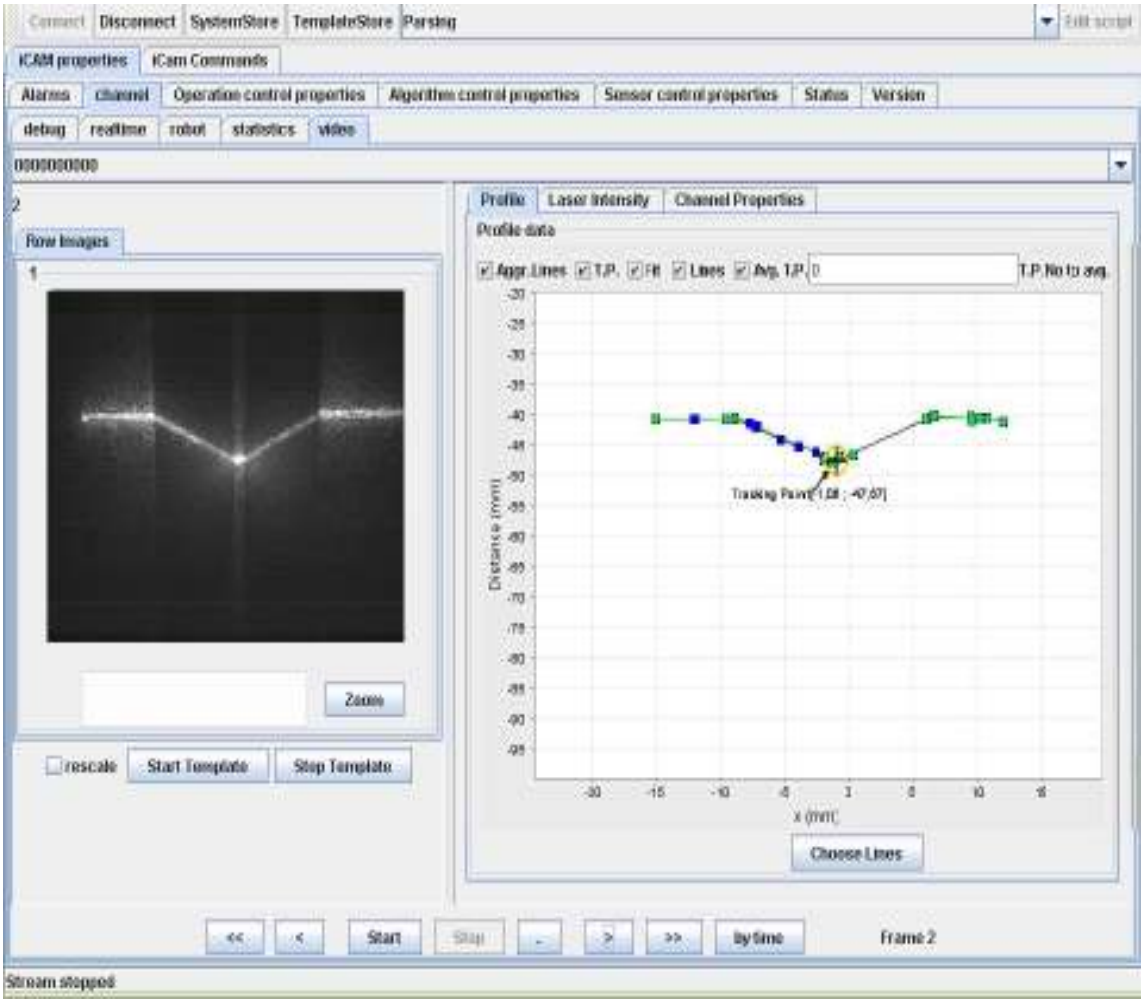
5.1. Работа с камерата за управление на заваръчен робот на фирмата KDS

Камерата на фирмата KDS съдържа множество от свойства, разположени йерархично, които се показват в таблици и табове по структурата описана по-горе. По-голям интерес за всяка система представляват потоковите данни и начина по който клиентът визуализира резултатите от тях. Камерата поддържа 5 такива потокови канала, които се визуализират в клиента като отделни табове, на едно ниво от йерархията. Клиентът предоставя за всеки един от каналите графичните секции за съответните типове данни, извлечени от системата.

Канали поддържани от системата на камерата на фирмата KDS:

- Видео канал – съдържа снимка с намалена резолюция за визуализиране и системна информация
- Канал за данни в реално време – съдържа DSP линейни сегменти, агрегирани сегменти, профил, заваръчна точки и измерени величини (включително разстояния и област на заваряване).
- Канал за диагностика – съдържа точки от функция за намиране на точки, цяла снимка
- Канал на робота – съдържа цялата информация изпратена през серийния канал към робота
- Канал за статистика – съдържа филтрирана информация за успех и неуспех при промяна на обект където неуспехите са разпределени между индивидуални ограничения.

Както е описано по-горе, всеки тип данни има определно място за визуализиране. Клиентът не знае предварително данните, които идват от канала, от който извлича потоковите данни, но снимките се визуализират на мястото за снимките, профилите, агрегирените сегменти, заваръчната точка и т.н. се визуализират в координатната система, останалите свойства се визуализират в таблици и т.н.



Фиг. 6 Работа с камера за управление на заваръчен робот на фирма KDS

На фиг. 6 е показан примерен резултат от извличането на данните от камерата. В лявата секция е снимката, идваща от камерата. В дясната част са резултатите от междинната обработка на устройството като профил, линейни сегменти и агрегирани линии. Там се вижда и крайния резултат от работата на системата – заваръчната точка (или точката, където е следващото местене на ръката на робота при заваряването). Точката е зададена с координати и се показва ясно с помощта на анотация. Цялата тази информация идва от системата и не е предварително известна. На фигурата се вижда, че клиентът адекватно и реално предоставя данните в удобен вид за потребителя.

5.2. Работа със система за микроизмервания разработвана в Химически факултет на СУ „Св. Кл. Охридски”.

Клиентското приложение е тествано със системата за микроизмервания, която се разработва в химическия факултет на СУ „Св. Кл. Охридски”. Тестовете показват, че клиентското приложение работи добре и с тази система като отговаря на нейните изисквания и предоставя наличната информация за системата. Системата за микроизмервания е в етап на развитие, поради което реалната работа на клиента не може да бъде напълно разгърната. До момента работата с нея показва, че клиентското приложение се държи адекватно спрямо нейните промени и етапи на развитие.

Заключение и насоки за бъдещо развитие

Технологиите в днешната динамична среда на компютърните системи се разрастват бързо и ежедневно навлизат все повече в човешките дейности. Това налага адекватен и удобен начин за достъп и контрол до системите във всяка една сфера на работа. Разработването на клиенти, отговарящи за конкретните нужди на дадена система не покрива тези бързи темпове на развитие, понеже изисква време и средства. В настоящата дипломна работа е разгледан един клиент, който може да бъде използван за всяка една система, независимо от това какви данни обработва тя и коя сфера от човешките дейности обслужва. На базата на описаните характеристики на приложението в изложението е показана неговата приложимост и всеобхватност. Основен акцент се поставя на факта, че клиентът реално не знае с каква система работи. Клиентът е тестван с две напълно различни и несъвместими системи. Описан е графичния интерфейс на клиента, който дава добра представа за системата, предоставя достъп до нейните данни и възможността за промяна и корекции в работата на системата. Тъй като основния акцент на дипломната работа не пада върху конкретните методи на визуализиране на данните, а върху представата, която се дава за системата чрез визуализирането им и начина им на организация, то това предлага усъвършенстването и развитието на клиента. Насоките за бъдещо развитие могат да бъдат в следните направления:

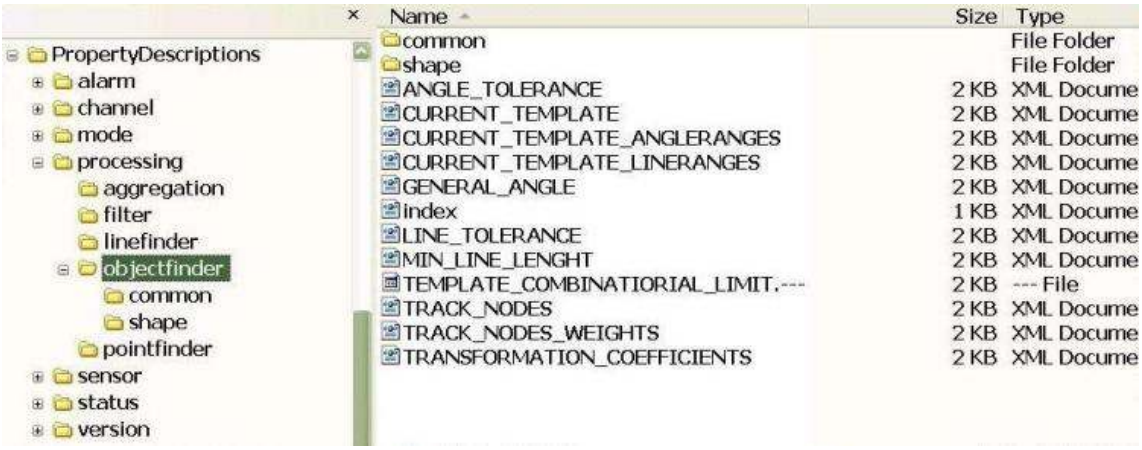
- реализация на визуализацията на тримерни данни. Такива данни реално могат да идват от почти всяка система в днешни дни.
- описанието на цветовете, прозорците, табовете и т.н. при визуализирането на двумерните данни може да бъде изваден от конфигурационния файл на клиента. Това например може да бъде изместено в самия графичен интерфейс и да се помисли за даване на възможност на потребителя да избира тези компоненти за визуализиране на различните данни.
- Описанието на командните бутони може да не е в конфигурационен файл. Добър подход би било да се дава възможност на потребителя да конфигурира командите, които иска да изпълни и да добавя или маха командни бутони. Това може да бъде разрешено или забранено за потребители с различни права на достъп до системата.

Приложение 1. Работа в offline режим

Клиентското приложение предоставя възможността потребителя да работи и в offline режим без да се свързва директно към системата. Това се постига ако локално на компютъра където се стартира клиентското приложение се копират данните от системата за нейната структура и свойства – т.е. директорийната структура на системата както и стойности на свойствата и пакети с данни от определен момент на работата на системата.

Йерархичната структура при работа в offline режим представлява йерархия от директории и описани в тях свойства чрез XML файлове. Тези XML файлове имат определена структура. Всяко свойство представлява един XML файл, който съдържа предварително дефинирана информация, необходима на протокола за комуникация между клиента и системата, за правилното анализиране и визуализиране на данните идващи от камерата. Протоколът за комуникация извлича информацията за от дефинициите на свойствата, която е необходима на клиента за визуразлизирането им.

Йерархичната структура на свойствата е представена като дърво, съдържащо директориите и XML файловете.



Фиг. 7 Йерархична структура на свойствата в измервателната камера на фирмата KDS

Всяка папка е описана с index.xml файл, намиращ се в папката, и се представя като таб в клиента, който съдържа информация за свойствата (файловете) и вътрешните папки на тази папка. Всяко свойство е описано в xml файл с неговото име. Свойствата от една папка се представят в таблиците за свойствата на системата. Имената, с които

се визуализира всеки един компонент на тази йерархична структура са описани в съответните xml файлове.

Примерен xml файл за свойство:

```
<?xml version="1.0" encoding="UTF-8" ?>
<propertydefinition location="/processing/objectfinder/">
  <Name value="/processing/objectfinder/LINE_TOLERANCE" />
  <pictogram value="" />
  <PropertyInternalName value="processing_objectfinder_line_tolerance" />
  <Label value="Template line tolerance" />
  <OnTypingValidator value="" />
  <OnSetValidator value="" />
  <!-- <PropertyType value="one of longarray, long ,floatarray, float , stringarray , string, enumeration, image,
  blob"/>  -->
  <PropertyType value="float" />
  <PropertyGroup value="template" />
  <!-- template|system -->
  <isReadOnly value="true" />
  <isAdmin value="false" />
  <Info value="" />
  <PropertyAllowedValues value="" />
  <!-- comma separated list, mandatory for enumerations -->
  <MinValue value="" />
  <MaxValue value="" />
  <Dependency value="" />
  <!-- list of properties which needs to be reread when this property changes  -->
  <isAdjustableByTheSystme value="true" />
  <arrayColumns value="" />
  <messageToBeGeneratedOnSet value="" />
  <commandToBeExecutedOnSet value="" />
</propertydefinition>
```

Това е добавено заради възможността потребителят да се съсредоточи върху точно определен и интересуващ го момент от работа на системата, запазен и съхранен локално, без да се налага спирането на нейната работа. За целта в клиентското приложение е добавен клас, наследник на интерфейса на комуникационния протокол – DirectoryServer. В него са реализирани методите на интерфейса, така че директорийната структура да бъде извлечена локално от системата от определено посочено място. Работата на класа е да симулира реалната работа на протокола и да предоставя необходимата информация за приложението като я извлича от това, което е съхранено на локалния компютър. За да работи потребителя в offline режим или реално да се свързва към система, при натискане на Connect бутона, му се дава възможност да избере начина на работа и да подаде съответно локален път до директрийната структура или адрес и порт към системата с която иска да работи. Това е още едно доказателство за универсалната работа на клиента и факта, че той реално не се интересува какво точно система стои зад комуникационния протокол.

Използвана литература

1. Soap tutorial (http://www.w3schools.com/soap/)

2. Overview of CORBA (http://www.cs.wustl.edu/~schmidt/corba-overview.html)

3. Why I'm using SOAP (http://www.ibm.com/developerworks/library/x-soapbx1.html)

4. Latest SOAP versions (http://www.w3.org/TR/soap/)

5. Apache Axis2/Java - Next Generation Web Services (http://ws.apache.org/axis2/)

6. Java Foundation Classes in a Nutshell September 1999 by David Flanagan published by O'Relly &Associates, Inc., 101 Morris Street, Sebastopol, CA 95472

7. Java in a Nutshell, Third Editon by David Flanagan November 1999 published by O'Relly &Associates, Inc., 101 Morris Street, Sebastopol, CA 95472

8. The JFreeChart Class Library Version 1.0.2, written by David Gilbert March 10, 2006

9. ImageJ (http://rsb.info.nih.gov/ij/)