Remedy IT

Your challenge - our solution

AXCIOMA

An eXtendable Component based Interoperable Open Model driven Architecture

AXCIOMA

An eXtendable Component-based Interoperable Open Model-driven Architecture

The Component Framework for Distributed, Real-Time, and Embedded Systems

AXCIOMA: the component framework for distributed, real-time, and embedded systems

Remedy IT

Your challenge - our solution

- **AXCIOMA** is the component technology enabling the **Industrial Internet of Things** (**IIoT**)
- The concrete communication middleware between components is a **deployment decision** and does not impact business logic
- **AXCIOMA** integrates multiple communication transports out of the box and more transports can be easily added
- **AXCIOMA** delivers portability and interoperability for **IIoT** applications through a standardized component model
- For more information take a look at our website https://www.axcioma.org/

What is AXCIOMA?

- AXCIOMA is a comprehensive software suite combining several Object Management Group (OMG) open standards
  - LwCCM, DDS, DDS4CCM, AMI4CCM, CORBA, IDL, IDL2C++11, and D&C
- AXCIOMA is based on
  - Interoperable Open Architecture (IOA)
  - Component Based Architecture (CBA)
  - Service Oriented Architecture (SOA)
  - Event Driven Architecture (EDA)
  - Model Driven Architecture (MDA)

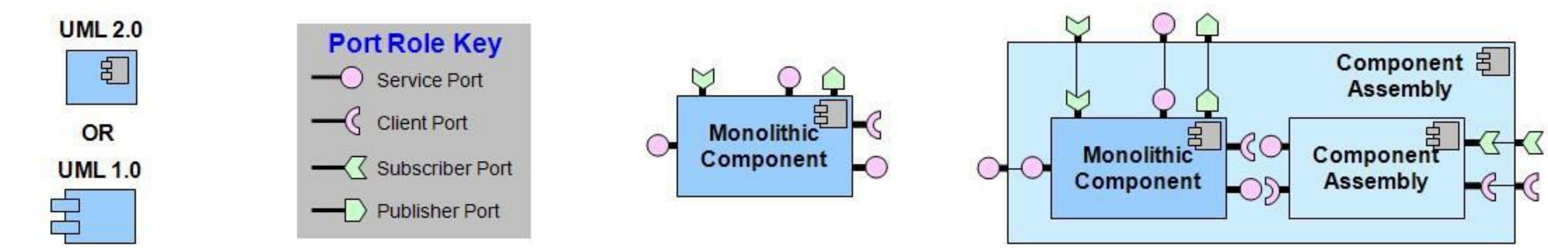
AXCIOMA

- AXCIOMA supports the design, development, and deployment of a distributed component based architecture
- A component based architecture encapsulates and integrates the following mechanisms in a “container”
  - Threading model
  - Lifecycle management
  - Connection management

Remedy IT
Your challenge - our solution

What is a Component?

- Independent revisable unit of software with well defined interfaces called “ports”
- Able to be packaged as an independently deployable set of files
- Smallest decomposable unit that defines standard ports is called a “monolithic component”
- A “component assembly” is an aggregation of monolithic components or other component assemblies



*A basic conceptual UML **Component**…* *… plus standard component **Port** types…* *… combine to define a CCM+DDS application **Monolithic Component*** *A **Component Assembly** defines a hierarchy of monolithic components and other assemblies*

Why Component Based Development? (1)

- Modularity
  - Components can be independently updated or replaced without impacting the rest of a system
- Reuse
  - Software is reusable at the component level instead of at the system level
- Interoperability
  - Well-defined ports and standards based development ensures interoperability between application components

Remedy IT

Your challenge - our solution

Why Component Based Development? (2)

- Extensibility
  - A Component Based Architecture (CBA) is inherently loosely-coupled, supporting easier extension of component and system functionality
- Scalability
  - Scalable from single component deployment to large distributed multi node deployments
- Reduced Complexity
  - Encapsulation, modularity, separation of concerns, and the establishment of hierarchical component dependencies contribute to reduced design & system complexity

Why Component Based Development? (3)

- Faster and Cheaper Development
  - Shorter design times, more reuse and less complexity
  - Faster time-to-market, faster software development
  - Focus changed to composition of a software-intensive system vs. all new design
  - Lower maintenance costs
- Quality & Reliability
  - Reuse and test/maintenance at the component level vs. at a monolithic system level

Remedy IT
Your challenge - our solution

Advantages of using Open Standards based API

- Open APIs are less prone to technology obsolescence
- No vendor lock-in
- Typically well vetted and designed
- Reuse of existing off-the-shelf technology
  - Implementations
  - Tools
  - Documentation
  - Training

AXCIOMA Design & Deployment

Remedy IT
Your challenge - our solution

- AXCIOMA clearly separates design, implementation, and deployment phases
- Components are designed to be location independent and communication middleware agnostic
- Components are implemented and tested in a highly decoupled fashion
- Deployment planning happens separately based on the complete system requirements
- The AXCIOMA framework handles the lifecycle for all components at runtime

Remedy IT
Your challenge - our solution

MDE Tooling

- MDE tools that support AXCIOMA
  - Zeligsoft CX for CBDDS
  - PTC Integrity Modeler IDL Profile
  - CoSMIC
  - Remedy IT Eclipse plugins
- MDE tools provide support for
  - Data modeling
  - Component modeling
  - Deployment modeling
  - Auto generation of IDL artifacts
  - Auto generation of D&C deployment plans

Interface Definition Language

- AXCIOMA uses Interface Definition Language (IDL) to define the type system, interaction patterns, and components
- IDL is vendor, programming language, and platform agnostic
- IDL is transformed to a programming language according to a so called language mapping
- Standardized language mappings exist for various programming languages, including C++, C++11, C, Java, Ruby, and others

IDL to C++11

- AXCIOMA supports the IDL to C++11 language mapping
- IDL to C++11 reuses as much as possible from the C++11 standard features
- The IDL generated types and support classes use C++11 features to provide a safe and easy to use API
- Business logic does not need to use C++11 language features

Generic Interaction Support

- Generic Interaction Support (GIS) enables the definition of generic interaction patterns
- Business logic uses interaction patterns to exchange information in a generic way
- Connectors realize a specific interaction pattern
- GIS allows the encapsulation of communication middleware, legacy systems, and hardware inside a connector
- Combining business logic and connectors is a deployment time decision, not an implementation decision

Remedy IT

Your challenge - our solution

Event Interaction Pattern

- AXCIOMA supports an event interaction pattern using the Generic Interaction Support
- The event interaction pattern defines extended ports for the following roles
  - Basic many-to-many publish subscribe messaging
  - Event distribution with optional user defined data

State Interaction Pattern

- AXCIOMA supports a state interaction pattern using the Generic Interaction Support
- The state interaction pattern defines extended ports for the following roles
  - Distributed state management and access
  - Distributed database functionality with eventual consistency

DDS Based State and Event Interaction Patterns

- AXCIOMA provides an implementation of the state and event interaction patterns using DDS as communication middleware
- Clearly separates business logic from all low level DDS details
- DDS QoS configuration is done using XML QoS profiles and not hardcoded into the business logic
- DDS Security provides secure interaction between the components

Advantages of AXCIOMA compared to plain DDS

Your challenge - our solution

- AXCIOMA delivers the following advantages compared to plain use of DDS
  - Delivers a concrete architecture instead of a messaging protocol
  - The implemented abstraction layer delivers DDS vendor neutrality
  - Achieves improved interoperability between components through standardized interaction patterns
  - Delivers portability of components between various operating systems and compilers
  - Comprehensive application layer MDE tooling support hides the complexity of DDS entities

Remedy IT

Your challenge - our solution

Request/Reply Interaction Pattern

- Using the Generic Interaction Support AXCIOMA realizes the request/reply interaction pattern
- Support for synchronous and asynchronous invocations
- Delivered with a function style API
- Defined in IDL using operations with arguments and an optional return value
- The application code that uses this interaction pattern is unaware of how the interaction pattern is realized

Remedy IT

Your challenge - our solution

CORBA Based Request/Reply Interaction Pattern

- AXCIOMA realizes the request/reply interaction pattern using CORBA
- The request/reply interaction pattern supports synchronous and asynchronous invocations
- CORBA communication is realized using the connector framework
- CORBA is a mature middleware technology delivering a well optimized transport mechanism
- Can use various communication transports like IIOP, SSLIOP
- AXCIOMA uses TAOX11 as CORBA implementation

Remedy IT

Your challenge - our solution

DDS Based Request/Reply Interaction Pattern

- AXCIOMA will also realize the request/reply interaction pattern using DDS as communication middleware
- The DDS connector implementation will hide all implied DDS topics and glue code imposed by the RPC4DDS standard from the business logic
- DDS Security provides secure interaction between the components

Integration of 3rd Party Middleware and Hardware

Remedy IT

Your challenge - our solution

- 3rd party communication middleware, legacy systems, and hardware are shielded from the application developer using the GIS connectors
- Connectors hide all communication middleware and hardware details
- AXCIOMA delivers a flexible framework for implementing custom connectors and code generators
- AXCIOMA supports the definition and implementation of user defined interaction patterns between components

Remedy IT

Your challenge - our solution

Execution Models

- AXCIOMA provides a single threaded and reentrant execution model as default execution model
- AXCIOMA will support the following additional execution models
  - Single threaded and non-reentrant
  - Multi threaded (thread pools)

Deployment using DnCX11

- AXCIOMA contains DnCX11 as a deployment tool supporting various deployment options
  - Centralized and decentralized deployment using D&C compliant tools
  - XML based and binary D&C compliant deployment plans
  - Easy to create text based deployment configuration files
  - Domain, node, and process as multiple levels of deployment

DnCX11 Single Node Deployment

- Deployment of one node using a node launcher
  - No need for a domain centralized and synchronized deployment
  - Nodes can be launched and torn down independently
  - Locality managers can be deployed as separate process or in-process with the node launcher
  - Components, connectors, and connections can be deployed using a very simple text based configuration file
  - Support for binary and XML D&C compliant deployment plans

DnCX11 Single Locality Deployment

- DnCX11 allows fully decentralized deployment of a single locality
- A locality represents one operating system process
- Support for binary and XML based D&C deployment plans and DnCX11 text based configuration files
- Using the static configuration support a single executable can be created containing AXCIOMA infrastructure and user components
  - Static deployment increases security and performance

DnCX11 Deployment Configuration Files

- DnCX11 has support for text based configuration files to
  - Configure deployment interceptors and handlers
  - Deploy components and connectors
  - Create local connections between components and connectors
  - Create remote connections between components

Remedy IT

Your challenge - our solution

AXCIOMA Advantages Compared to CIAO

- AXCIOMA supports the most features from CIAO and DAnCE
- AXCIOMA has the following advantages compared to CIAO
  - Much easier to use language mapping which increases the productivity of the programmer
  - Reduced application code
  - Up to 70% footprint reduction for your component related generated code
  - Support for regeneration of business logic without losing already implemented code
  - Prevents possible memory leaks or invalid memory access at runtime

AXCIOMA Advantages Compared to CIAO

Remedy IT

Your challenge - our solution

- And AXCIOMA has even more advantages
  - Simplified compilation of all IDL generated artifacts
  - The request/reply interaction pattern using CORBA is realized using the connector framework and not implicitly by the framework
  - Will realize the request/reply interaction pattern using DDS
  - Simplified and more powerful deployment tooling
  - Framework for implementing custom connectors
  - Extensible logging framework

Remedy IT

Your challenge - our solution

AXCIOMA Advantages Compared to DAnCE

- AXCIOMA has the following advantages compared to DAnCE as deployment tool
  - Improved configurability
  - Improved tool consistency
  - Extended deployment options (centralized, node, and locality)
  - Multiple options to support runtime debugging of component implementations

Want to know more about AXCIOMA?

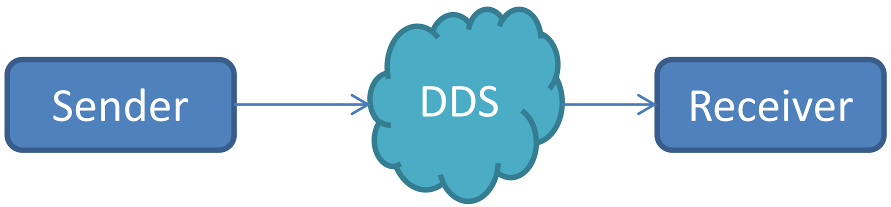
Remedy IT
Your challenge - our solution

- Contact Remedy IT at sales@remedy.nl
- Call us at +31-88-0530000
- Check our website at https://www.remedy.nl
- Check AXCIOMA at https://www.axcioma.org
- Follow us on Twitter @RemedyIT

Shapes AXCIOMA example

Example overview

- This example demonstrates 2 components exchanging data using the event interaction pattern
  - Sender writes shapes samples to a event connector
  - Receiver receives shapes samples from a event connector



Shape IDL definition

- In order to exchange data we define a publish subscribe message type in IDL which is used by all components
- Based on the IDL message type definition AXCIOMA will generate
  - C++11 type representation
  - C++11 DDS data reader and writer API
  - Conversion layer to integrate a specific DDS vendor

IDL Shapetype

Your challenge - our solution

IDL definition

```
struct ShapeType {
  string color; //@key
  long x;
  long y;
  long shapesize;
};
```

Component Definition

Remedy IT
Your challenge - our solution

- Two components are defined in this example, both use the GIS DDS4CCM extended ports
- An extended port delivers a specific interaction pattern
  - State: state based data exchange
  - Event: event based data exchange
- All extended ports are available by instantiating the DDS4CCM templated module with a concrete data type definition
  - `module CCM_DDS::Type <ShapeType, ShapeTypeSeq> ShapeType_conn;`

Component Definitions

Sender

```
module Shapes {
  component Sender {
    port ShapeType_conn::DDS_Write
      info_write;
  };
};
```

Receiver

```
module Shapes {
  component Receiver {
    port ShapeType_conn::DDS_Listen
      info_out;
  };
};
```

Sender implementation

Remedy IT

Your challenge - our solution

```
// Sender component class declaration and implementation which publishes one sample to DDS
class Sender_i : public IDL::traits<CCM_Sender>::base_type
{
public:
  // Setter method to receive the component context
  virtual void set_session_context(IDL::traits<Components::SessionContext>::ref_type ctx) override {
   context_ = IDL::traits<Shapes::CCM_Sender_Context>::narrow (ctx);
  }
  // Lifecycle callback indicating we have received our settings, register an instance to DDS
  virtual void configuration_complete () override {
    IDL::traits<Shapes::ShapeType_conn::Writer>::ref_type writer =
      context_->get_connection_info_write_data();
    instance_handle_ = writer->register_instance (square_);
  }
  // Lifecycle callback indicating we can start our functionality, write one sample to DDS
  virtual void ccm_activate () override {
    IDL::traits<Shapes::ShapeType_conn::Writer>::ref_type writer =
      context_->get_connection_info_write_data();
    writer->write_one (square_, instance_handle_);
  }
  // Lifecycle callback that we are going to shutdown, unregister the instance from DDS
  virtual void ccm_passivate () override {
    IDL::traits<Shapes::ShapeType_conn::Writer>::ref_type writer =
      context_->get_connection_info_write_data();
    writer->unregister_instance (square_, instance_handle_);
  }
  virtual void ccm_remove () override {}
private:
  IDL::traits<Shapes::CCM_Sender_Context>::ref_type context_;
  DDS::InstanceHandle_t instance_handle_;
  // Use C++11 uniform initialization to initialize the member
  ShapeType square {“GREEN”, 10, 10, 1};
};
```

Receiver implementation (1)

```
// Receiver component declaration and implementation which receives the samples from DDS
class Receiver_i : public IDL::traits<CCM_Receiver>::base_type
{
public:
  // Setter method to receive the component context
  virtual void set_session_context(IDL::traits<Components::SessionContext>::ref_type ctx) override {
   context_ = IDL::traits<Shapes::CCM_Receiver_Context>::narrow (ctx);
  }
  virtual void configuration_complete () override {}
  // Lifecycle callback indicating we can start our functionality, indicate we want sample by sample
  virtual void ccm_activate () override {
    IDL::traits<CCM_DDS::DataListenerControl>::ref_type lc =
      context_->get_connection_info_data_control();
    lc->mode (CCM_DDS::ListenerMode::ONE_BY_ONE);
  }
  virtual void ccm_passivate () override {}
  virtual void ccm_remove () override {}
  // Retrieve the facet executor that implements the listener functionality
  IDL::traits<Shapes::ShapeType_conn::CCM_Listener>::ref_type get_info_out_data_listener () {
    if (!data_listener_) data_listener_ = CORBA::make_reference<info_out_i> (context)_;
    return data_listener_; }
private:
  IDL::traits<Shapes::CCM_Sender_Context>::ref_type context_;
  IDL::traits<Shapes::ShapeType_conn::CCM_Listener>::ref_type data_listener_;
};
```

Receiver implementation (2)

Remedy IT

Your challenge - our solution

```
// Listener facet implementation, receives the sample from DDS and just dumps it to the console
class info_out_i: public IDL::traits<Shapes::ShapeType_conn::CCM_Listener>::base_type
{
public:
  info_out_i(IDL::traits<Components::CCM_Receiver_Context>::ref_type ctx) : context_ (ctx) {}
  // Callback to inform the component that a sample has been received by DDS
  virtual void on_one_data (const ShapeType& shape, CCM_DDS::ReadInfo&) override {
    std::cout << "Received " << shape << std::endl;
  }
  virtual void on_many_data (const ShapeTypeSeq&, CCM_DDS::ReadInfoSeq&) override {}
private:
  IDL::traits<Shapes::CCM_Sender_Context>::ref_type context_;
};
```

Remedy IT

Your challenge - our solution

Deployment

- The example components can be deployed using
  - Centralized deployment using the D&C compliant deployment tools
  - Deployment of one node using the single node launcher
  - Deployment of one process using the single locality launcher
    - Support for the D&C compliant deployment plans
    - Support for simple text based deployment configuration
    - Easy to start and use directly from a debugger

Remedy IT

Your challenge - our solution

Background slides

IDL to C++11 (1)

- The IDL to C++11 language mapping is a formal open standard created with the following goals
  - Simplify development compared to IDL to C++
  - Reduce amount of possible programming errors
  - Gain runtime performance
  - Reduce size of application code
  - Use C++11 standard types and constructs as much as possible

IDL to C++11 (2)

Remedy IT
Your challenge - our solution

- The specification is available from http://www.omg.org/spec/CPP11
- For background, details, tutorials, examples see
  - https://www.taox11.org/

TAOX11

- Compliant with IDL to C++11 v1.3
- Support for CORBA AMI
- New IDL compiler with front end supporting IDL2, IDL3, and IDL3+

Contact

Remedy IT
The Netherlands

tel.: +31(0)88 – 053 0000
e-mail: sales@remedy.nl
website: www.remedy.nl

Twitter: @RemedyIT
Slideshare: RemedyIT