Remedy IT

Your challenge - our solution

Integrating OpenDDS into AXCIOMA, the component approach

Johnny Willemsen (jwillemsen@remedy.nl)

CTO Remedy IT

https://www.remedy.nl

Remedy IT

Remedy IT

Your challenge - our solution

- Remedy IT is specialized in communication middleware and component technologies
- Strong focus on open standards based solutions
- Our customers are in various domains including telecom, aerospace and defense, transportation, industrial automation
- For more information take a look at our website https://www.remedy.nl

Remedy IT

Your challenge - our solution

What We Do

- Develop and support implementations of OMG open standards
  - TAOX11, AXCIOMA, TAO, CIAO, R2CORBA, JacORB, OpenDDS
- Deliver services related to OMG standards including the CORBA, CCM, and DDS standard
- Support various other open source projects including ACE

Remedy IT

Your challenge - our solution

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

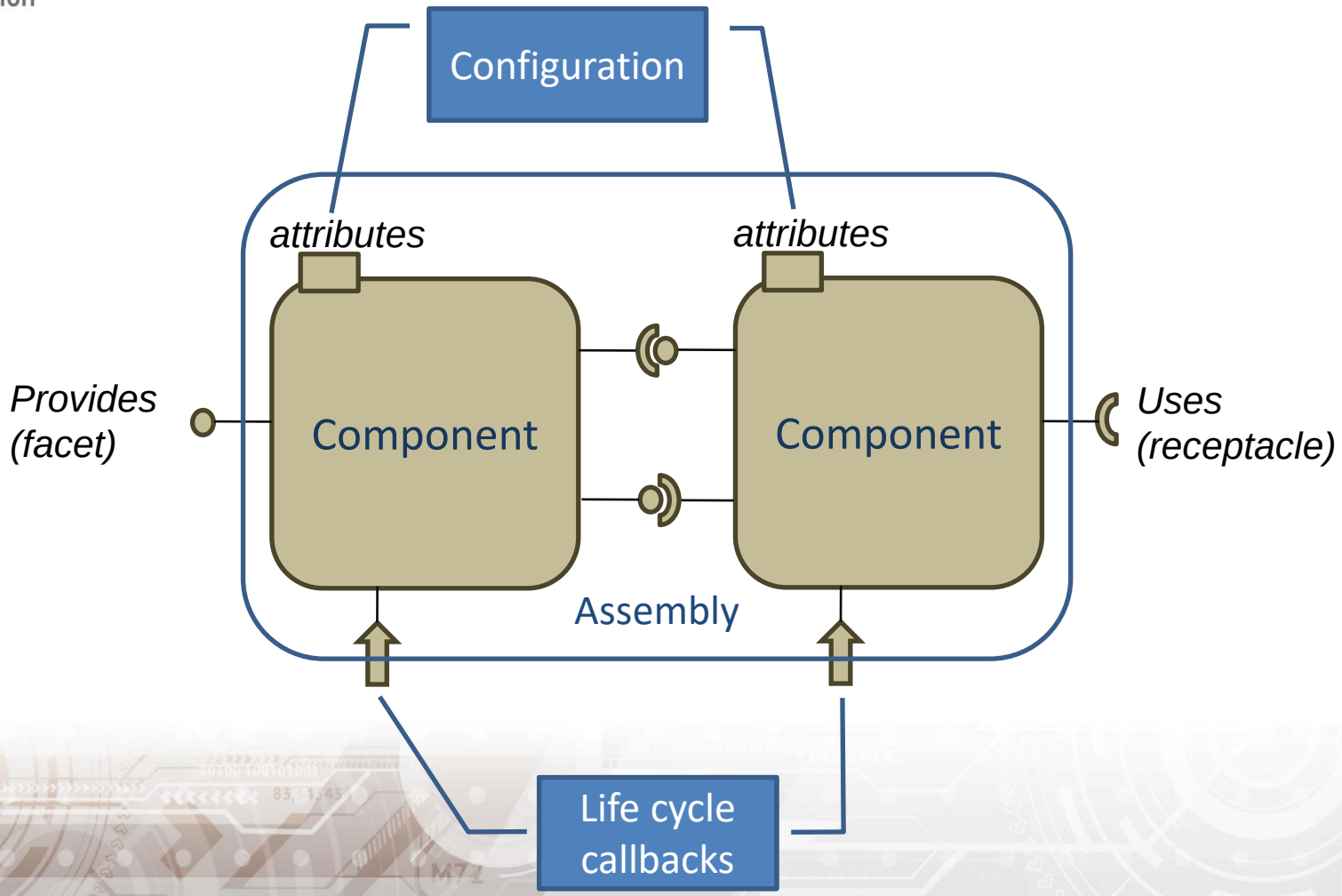
What Is a Component?

- Independent revisable unit of software with well defined interfaces, called ports
- Can be packaged as an independent 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

Component

Remedy IT

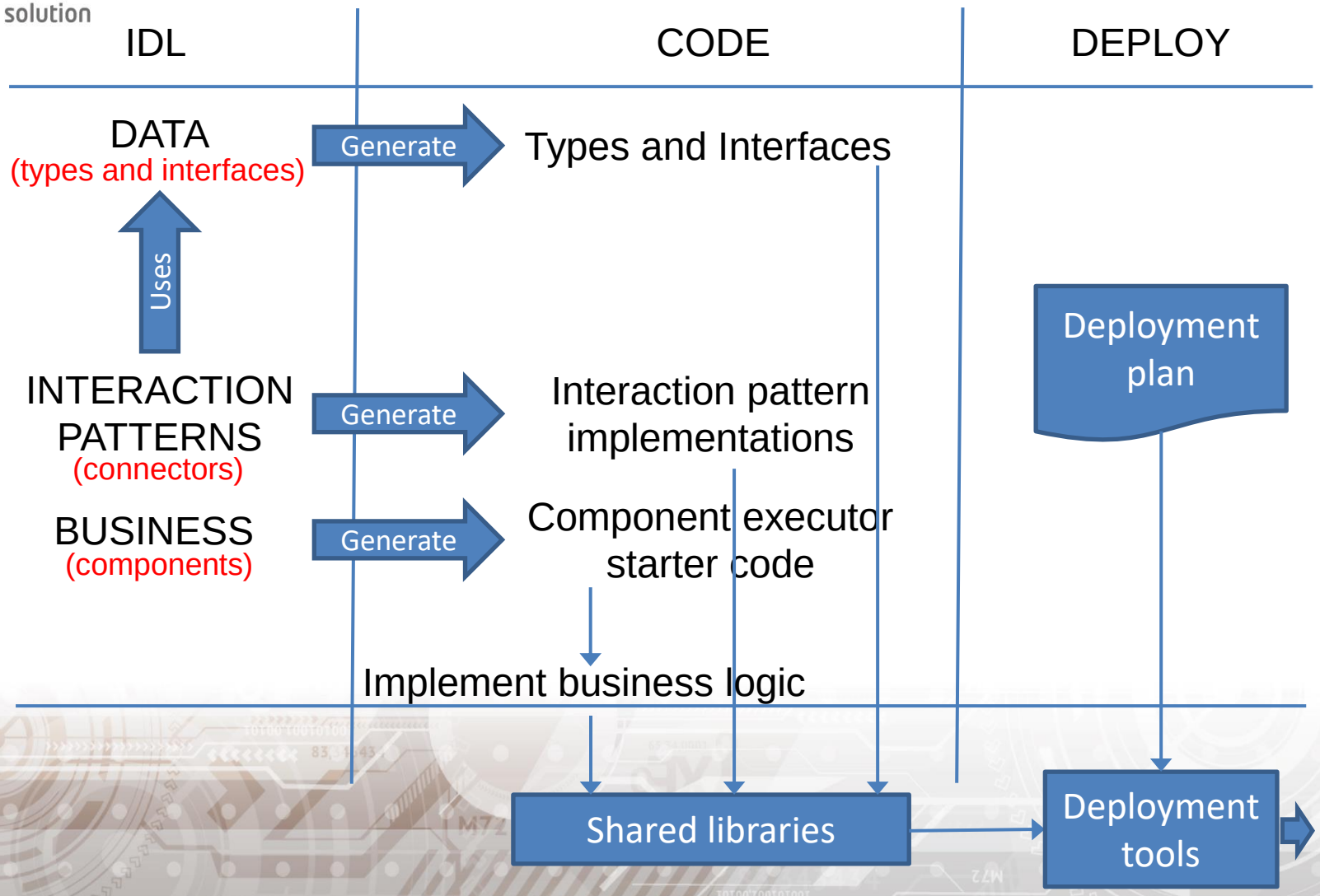
Your challenge - our solution



Component Framework

Remedy IT

Your challenge - our solution



Remedy IT

Your challenge - our solution

Interaction Patterns

- Define how components interact with the outside world
  - Request/Reply interaction
    - client, server, asynchronous client, and asynchronous server
  - Event interaction
    - supplier, push consumer, and pull consumer
  - State interaction
    - observable, passive observer, push observer, pull observer, and push state observer
- All these interaction patterns can be realized using DDS
- CORBA, ZeroMQ, and MQTT support available

Remedy IT

Your challenge - our solution

Our AXCIOMA DDS Challenge

- Integrate OpenDDS into AXCIOMA
- Provide the IDL to C++11 API to our users
- Abstract and optimize DDS through the interaction patterns
  - Request/reply
  - State (DDS4CCM)
  - Event (DDS4CCM)

IDL to C++11 Language Mapping (I)

- Simplified mapping for C++
  - Make use of the standard C++ library as much as possible
- Make use of the C++11 features to
  - Reduce amount of application code
  - Reduce amount of possible coding errors by providing a safer API
  - Gain runtime performance
  - Speedup development and testing
    - Faster time to market
    - Reduced costs
    - Reduced training time

Remedy IT

Your challenge - our solution

IDL to C++11 Language Mapping (II)

- An IDL interface maps to so called reference types
- Reference types are automatically reference counted
- A nil reference type is represented as `nullptr`
- A boolean operator for reference comparison is available
- Invoking an operation on a nil reference results in a `INV_OBJREF` exception, no need whether object references are valid throughout your business code
- Support for strong and weak references

Remedy IT

Your challenge - our solution

DDSX11

- OpenDDS currently does not fully support the IDL to C++11 language mapping (no DDS vendor supports this mapping completely)
- DDSX11 performs the bridging between the IDL to C++11 and OpenDDS C++ API
- Hides all vendor DDS API details from the programmer
- Combination of
  - IDL based code generation
  - C++11 code generation
  - Core support classes and templates
- Versioned namespaces are used to place OpenDDS types into their own distinct namespace

Remedy IT

Your challenge - our solution

DDSX11

- Improvements/enhancements contributed to OpenDDS during development of DDSX11 for OpenDDS
- QoS XML support
- Uses TAO 3.x (C++11 support mandatory)
- 100% open source solution enables easy porting and development
- OpenDDS DDSX11 support is mature and stable

Remedy IT

Your challenge - our solution

DDSX11 Conversion traits

- For DDSX11 the C++11 types are leading
- For each IDL defined type we provide a trait with helper methods to convert between C++ and C++11
  - Basic type traits are part of the DDSX11 core
  - Constructed type traits are generated by our RIDL IDL compiler
    - Ruby based IDL compiler
    - Generates traits for a specific DDSX11 vendor
- DDSX11 uses the traits and is unaware of the real data type

Remedy IT

Your challenge - our solution

DDSX11 Conversion traits

- Conversion traits are currently optimized for OpenDDS using the ‘old’ C++ API
- Traits can be generated differently for other DDS vendors
- At the moment the C++ and C++11 type are the same the conversion traits are optimized away by the compiler
  - DDSX11 and user code doesn’t need to be changed

Remedy IT

Your challenge - our solution

Optimizing DDS Usage

- DDS API is hidden from the programmer
- Knowledge about how DDS setup is part of the connector
- The DDS usage knowledge is implemented and optimized once
  - Usage of domain participants (how many)
  - Reuse of topics
  - Clean shutdown of DDS
- DDSX11 uses IDL4 annotations which are converted to the DDS vendor specific setting

Remedy IT

Your challenge - our solution

Component and DDS Execution Model

- Components run in a single threaded, re-entrant environment
- Callbacks from DDS threads are dispatched onto our main component thread
- No locking in user code necessary
- Additional execution models are available for more complex execution environments

Remedy IT

Your challenge - our solution

Testing

- All our connector and framework functionality is tested automatically
- No need for special DDS test connectors
- Special test components that trigger fault conditions
  - Sometimes need to be combined with specific QoS settings
- Keep QoS and configuration as simple as possible

Shapes Example

Remedy IT

Your challenge - our solution

Remedy IT

Your challenge - our solution

Generated ShapeType Class

```
class ShapeType
{
public:
  ShapeType () = default;
  ~ShapeType () = default;
  ShapeType (const ShapeType&) = default;
  ShapeType (ShapeType&&) = default;
  ShapeType (color_type color, int32_t x, int32_t y, int32_t shapesize);
  ShapeType& operator= (const ShapeType&) = default;
  ShapeType& operator= (ShapeType&&) = default;
  …
  // Getters and Setters
private:
  // Struct members as private members
};

ShapeType shape {"GREEN", 0, 0, 15 };
std::cout << "Created ShapeType " << shape << std::endl;
ShapeType shape1 = shape;
ShapeType shape2 (shape1);
```

Remedy IT

Your challenge - our solution

Component Executor Class

```
/// Component Executor Implementation Class : Publisher_comp_exec_i
class Pubisher_comp_exec_i final
  : public virtual IDL::traits<CCM_Publisher_comp>::base_type
{
public:
  /// Constructor
  Publisher_comp_exec_i ();

  /// Destructor
  ~Publisher_comp_exec_i () override;

  /** @name Component port operations. */
  //@{
  /// Factory method and getter for the control facet
  /// @return existing instance of facet if one exists, else creates it
  IDL::traits<Shapes::CCM_Control>::ref_type get_control () override;
  //@}
  ...
```

Remedy IT

Your challenge - our solution

Facet Executor Class

```
Shapes::ReturnStatus
control_exec_i::setLocation (uint16_t x, uint16_t y)
{
  Shapes::ReturnStatus status = Shapes::ReturnStatus::RETURN_ERROR;

  auto cex = IDL::traits<Publisher_comp_exec_i>::narrow (
    this->component_executor_.lock ());

  if (cex)
    status = cex->setLocation (x, y);
  else
    std::cout << "setLocation - failed to lock executor." << std::endl;

  return status;
}
```

Remedy IT

Your challenge - our solution

Write a DDS sample

```
// Get the writer port which we use to write a DDS sample
IDL::traits< ::Shapes::ShapeType_conn::Writer>::ref_type writer =
        this->context_->get_connection_info_write_data ();

// Write one sample square for the given instance handle
writer->write_one (this->square_, this->instance_handle_);
```

Remedy IT

Your challenge - our solution

Receive a DDS sample

```
// Data is delivered through a callback
void
info_out_data_listener_exec_i::on_one_data (
    const ::ShapeType& datum,
    const ::CCM_DDS::ReadInfo&)
{
  std::cout << "Received " << datum << std::endl;
}
```

Remedy IT

Your challenge - our solution

Conclusion

- DDS fits perfect into a component based approach
- DDSX11 abstracts vendor differences and improves portability of user code
- OpenDDS is a mature and stable supported DDS implementation for DDSX11
- IDL to C++11 simplifies user code, increases performance, and reduces time to implement

Remedy IT

Your challenge - our solution

Contact

Remedy IT
The Netherlands

e-mail: sales@remedy.nl
website: www.remedy.nl