



Remedy IT

Your challenge - our solution



LONDON CONNEXT CONFERENCE 2015



Connecting Our Users: The DDS Experts. — October 14-15, 2015

Integrating DDS into AXCIOMA, the component approach

Johnny Willemsen (jwillemsen@remedy.nl)

CTO Remedy IT

<https://www.remedy.nl>



Remedy IT

Your challenge - our solution

Remedy IT



- Remedy IT is specialized in communication middleware and component technologies
- Strong focus on open standards based solutions
- Actively involved in the Object Management Group, chairing several OMG standardization efforts
- 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



- Global Service Delivery Partner for RTI Connex DDS
- Develop implementations of OMG open standards
 - Open source: TAOX11, AXCIOMA, TAO, CIAO, R2CORBA
- Deliver services related to OMG standards including the CORBA, CCM, and DDS standard
- Develop open standards as part of the Object Management Group



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)



Remedy IT

Your challenge - our solution

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



What Is a Component?



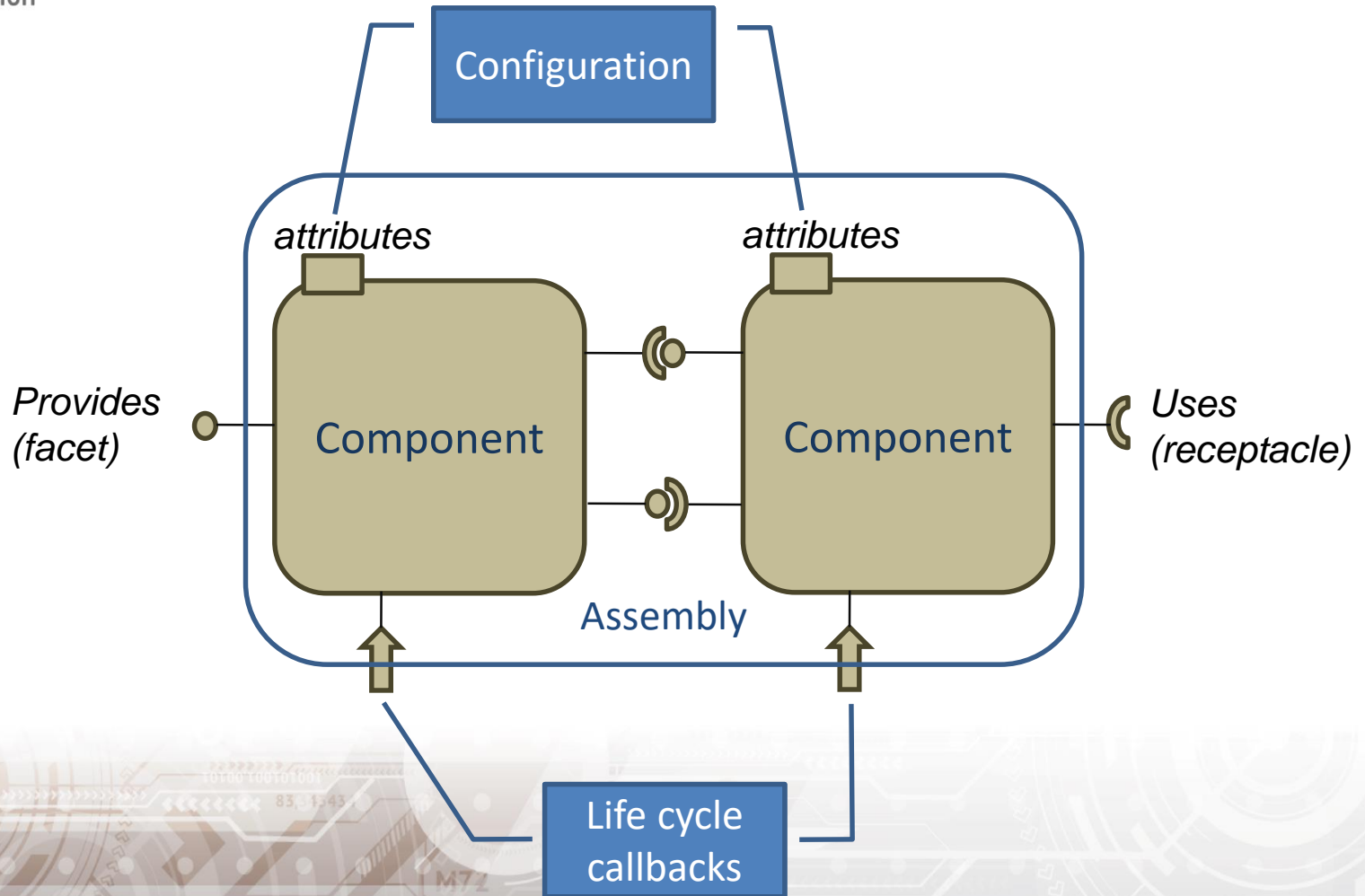
- Independent revisable unit of software with well defined interfaces, called ports
- Can be packaged as an independent deployable set of files
- Smallest decomposable unit that defines standard ports is called a monolithic component
- An component assembly is an aggregation of monolithic components or other component assemblies



Remedy IT

Your challenge - our solution

Component



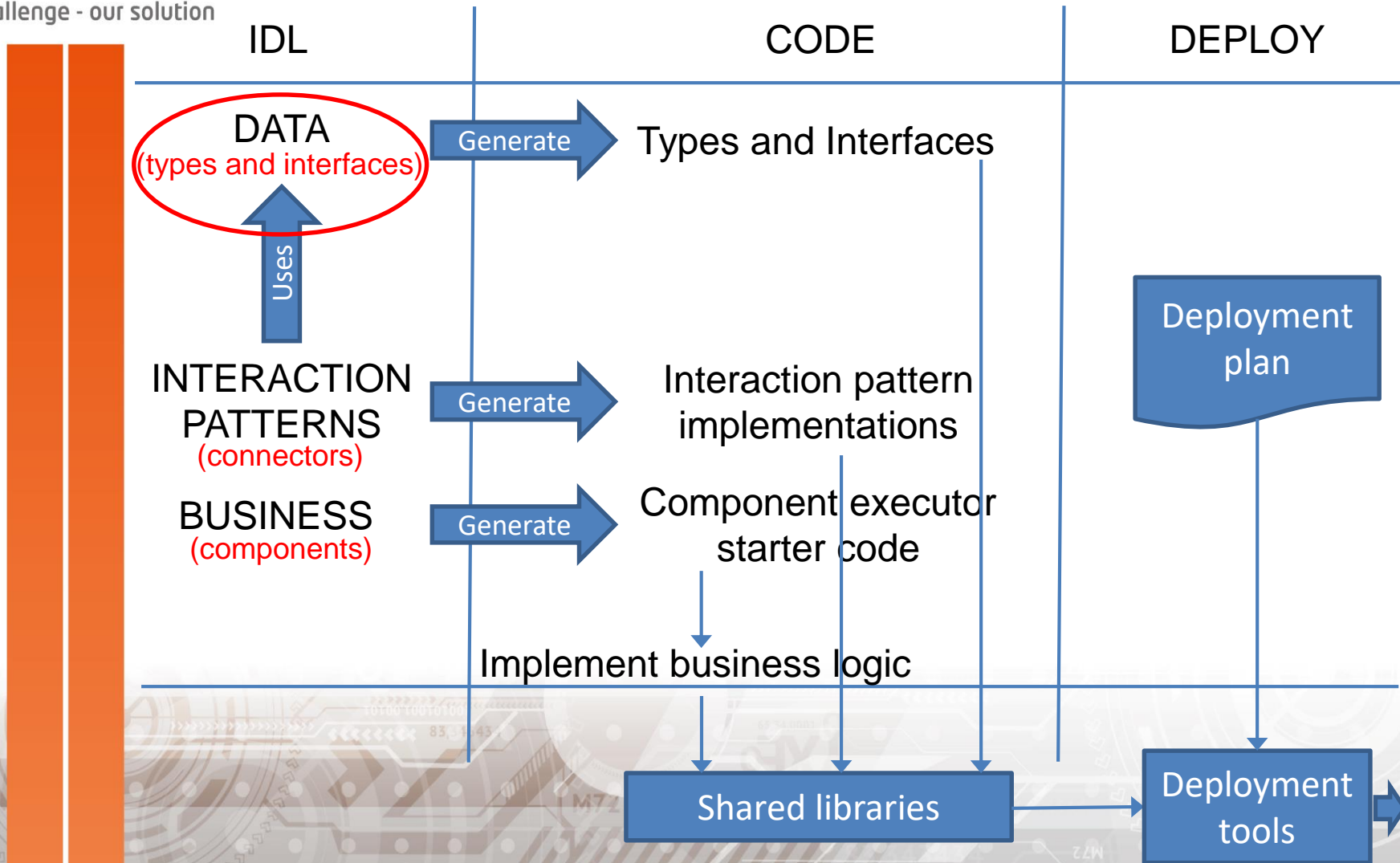


Component Framework



Remedy IT

Your challenge - our solution

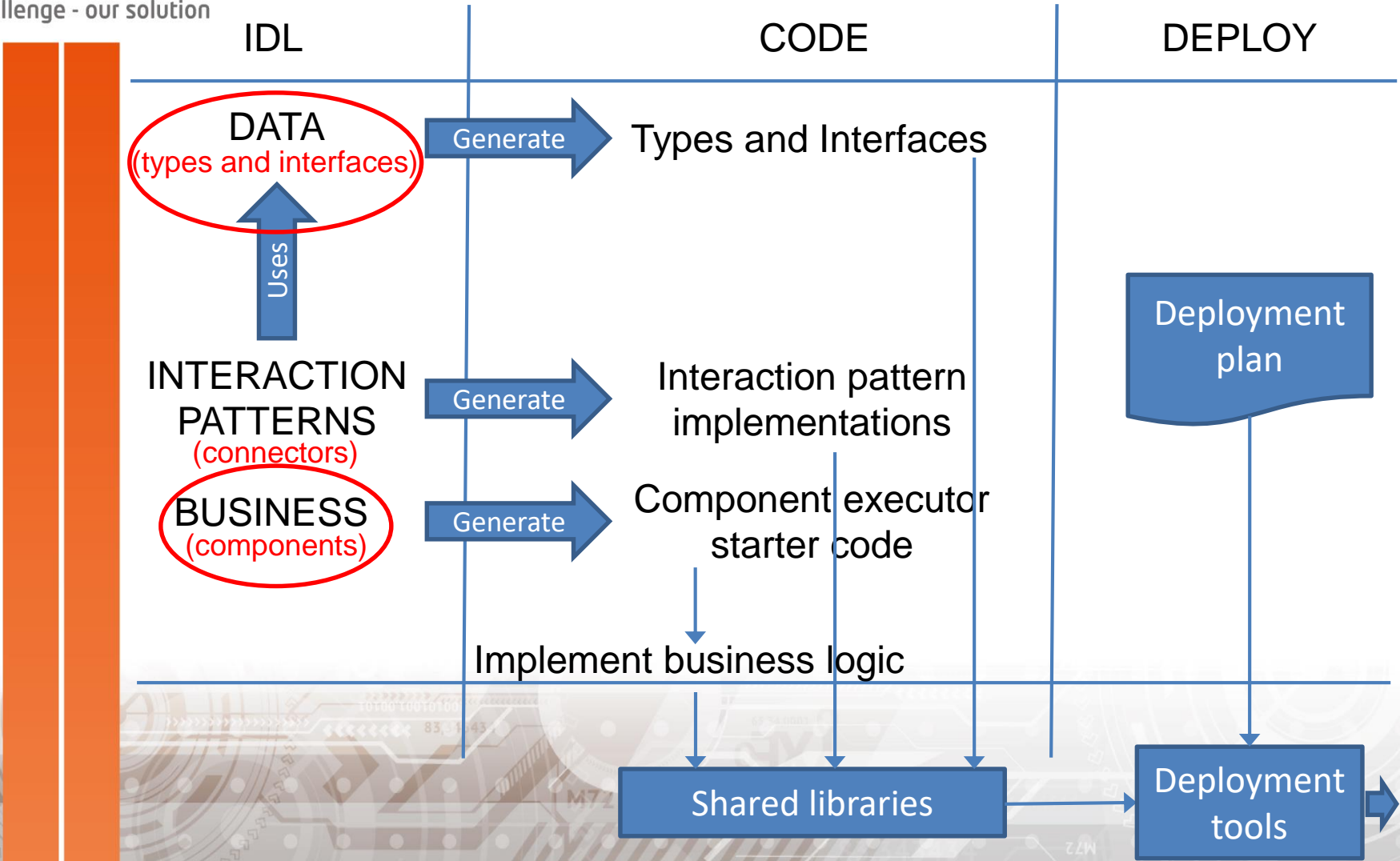




Remedy IT

Your challenge - our solution

Component Framework

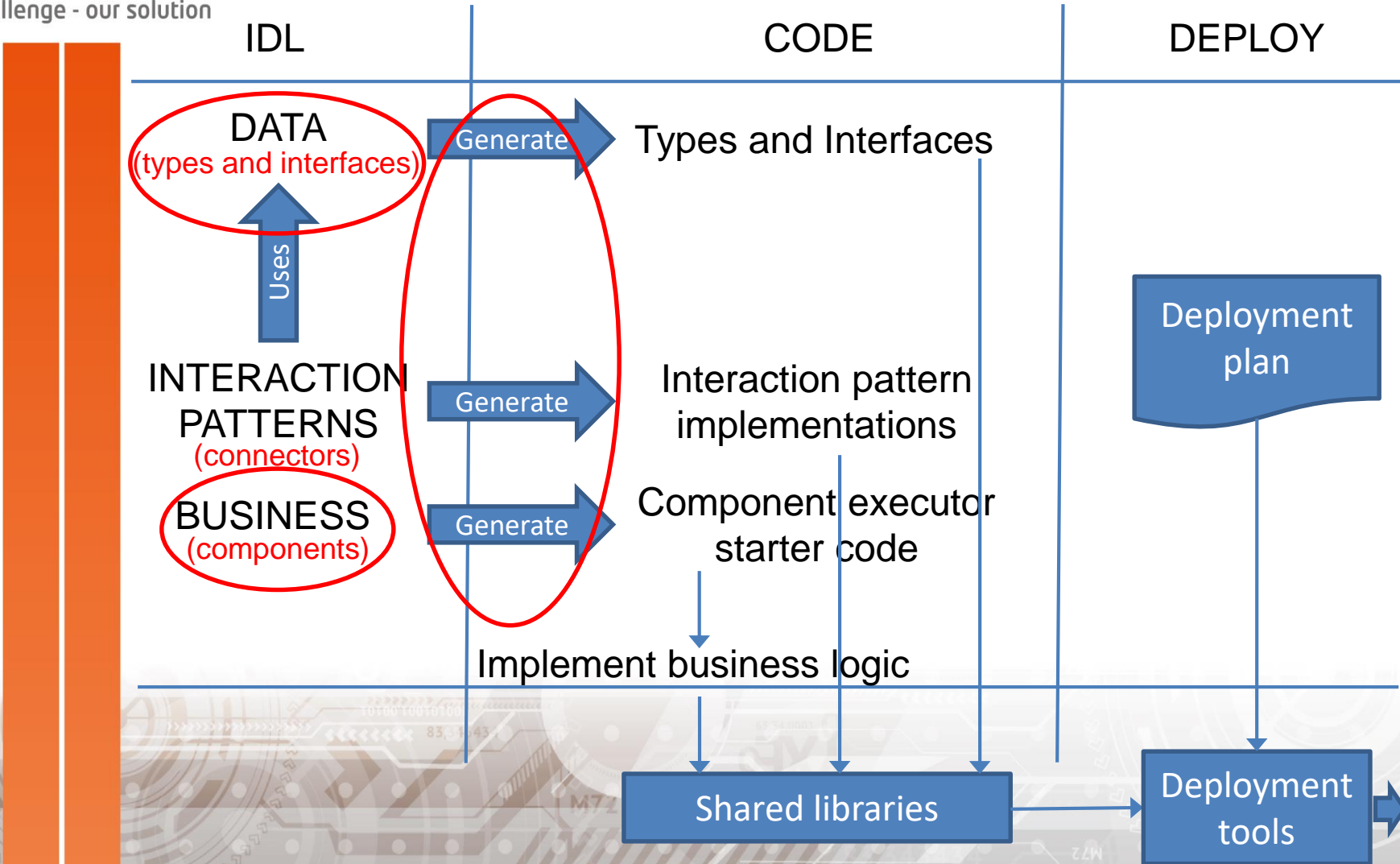




Remedy IT

Your challenge - our solution

Component Framework



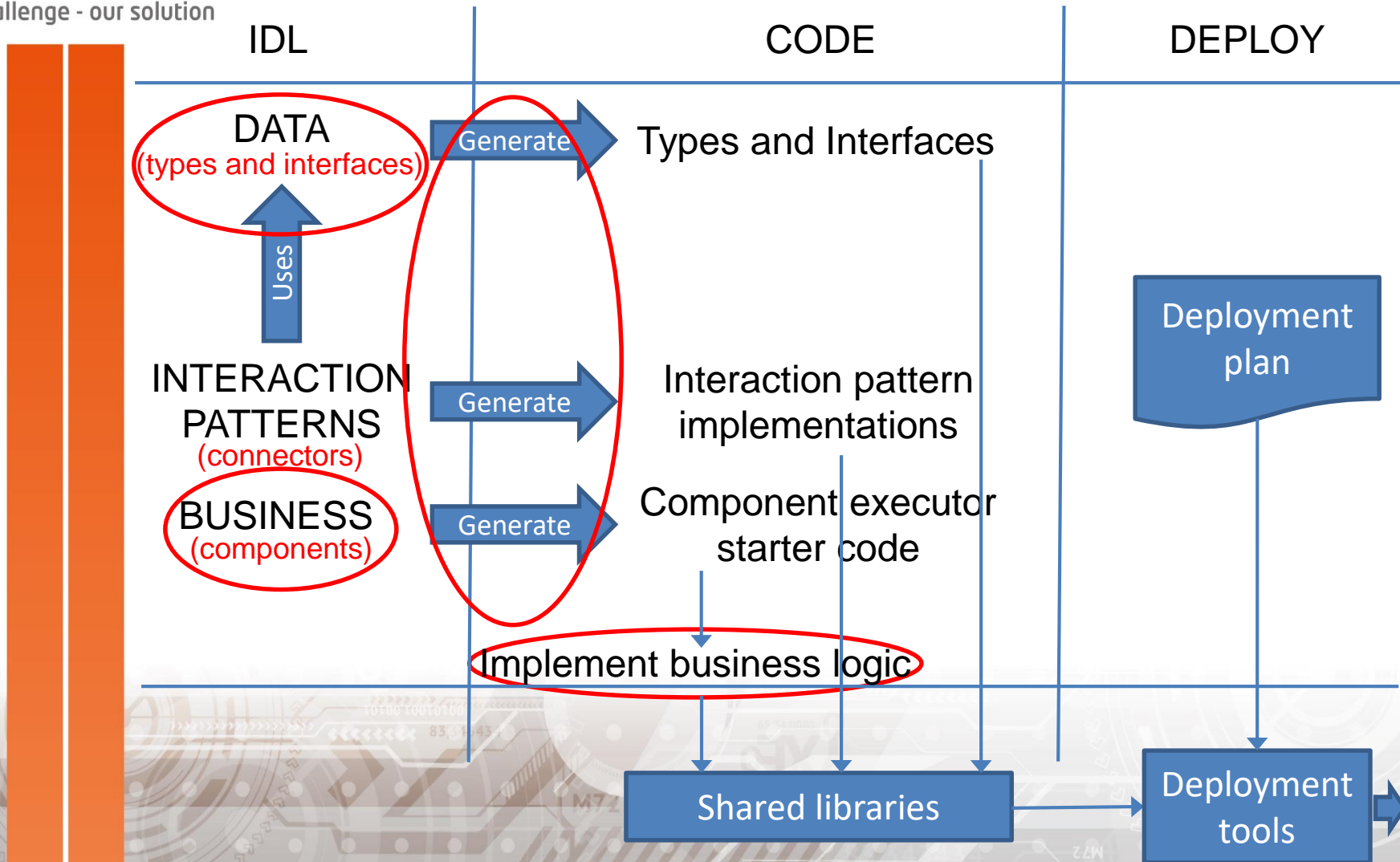


Component Framework



Remedy IT

Your challenge - our solution



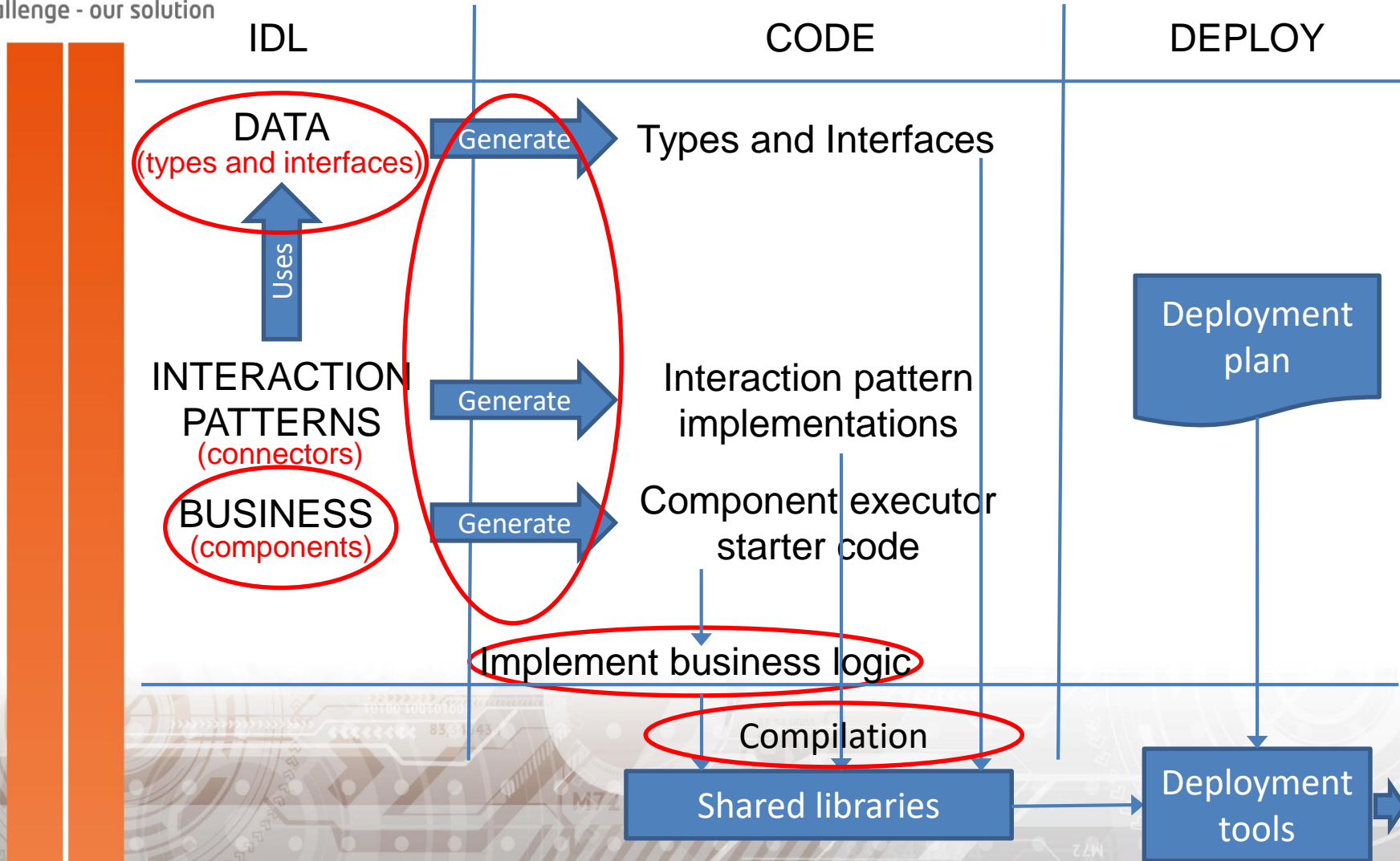


Component Framework



Remedy IT

Your challenge - our solution



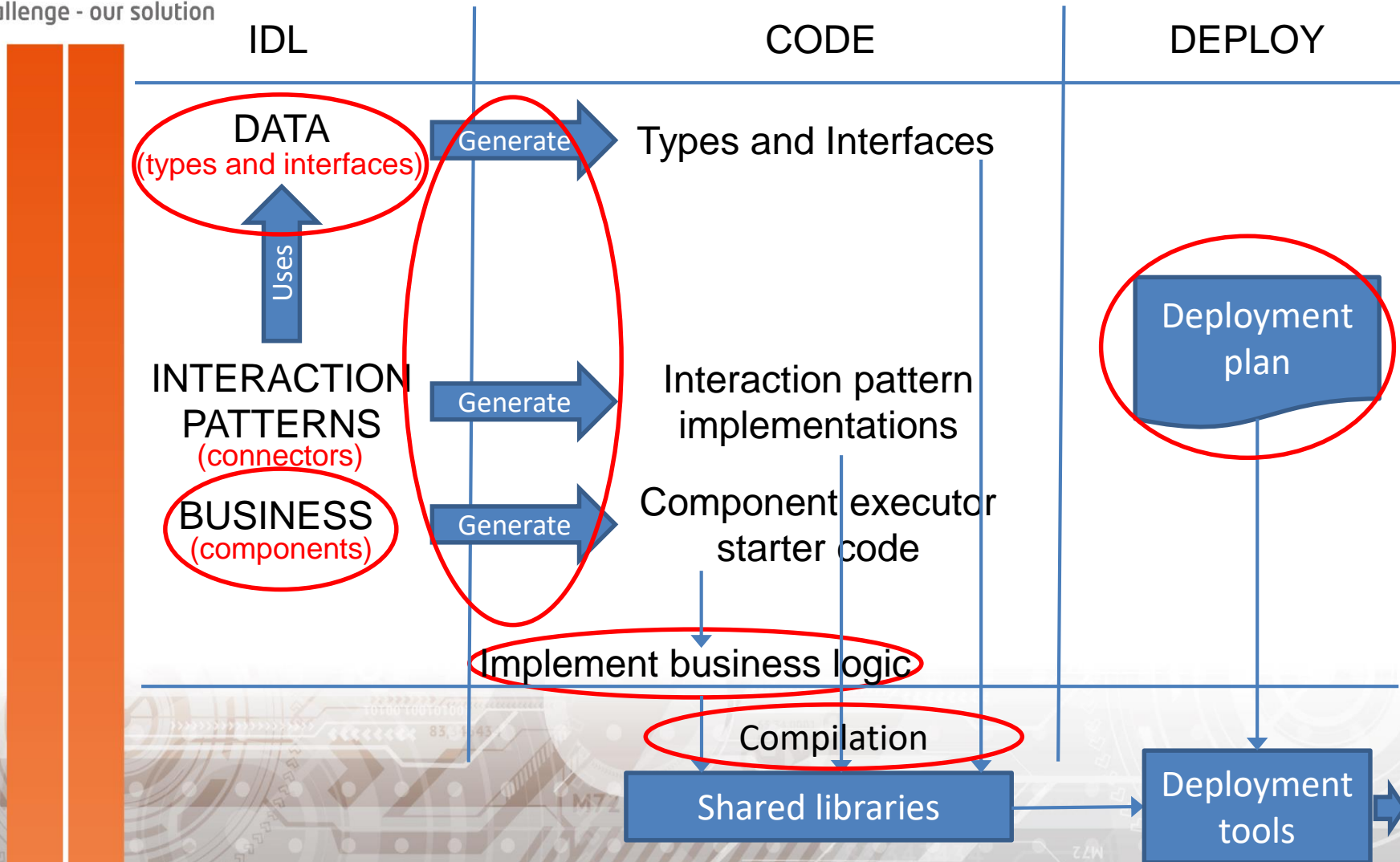


Component Framework



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



Remedy IT

Your challenge - our solution

Our AXCIOMA DDS Challenge



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



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



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



DDSX11



- RTI Connex DDS currently does not support the IDL to C++11 language mapping
- DDSX11 performs the bridging between the IDL to C++11 and RTI Connex DDS C++ API
- Hides all vendor API details from the programmer
- Combination of
 - IDL based code generation
 - C++11 code generation
 - Core support classes and templates



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 core
 - Constructed type traits are generated by our RIDL IDL compiler
 - Generated for a specific vendor
- DDSX11 uses the traits and is unaware of the real type



DDSX11 Conversion traits



- Conversion traits are currently optimized for RTI Connex DDS using the 'old' C++ API
- Traits can be generated differently for other vendors or a different RTI version
- 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



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 can use IDL4 annotations which are converted to the DDS vendor specific setting



Component and DDS Execution Model



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



Testing



- All our connector and framework functionality has to be 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
- On heavy loads sometimes DomainParticipant discovery is missed
 - Wait on DDS callbacks like `publication_matched` before starting the real test code
- Keep QoS and configuration as simple as possible

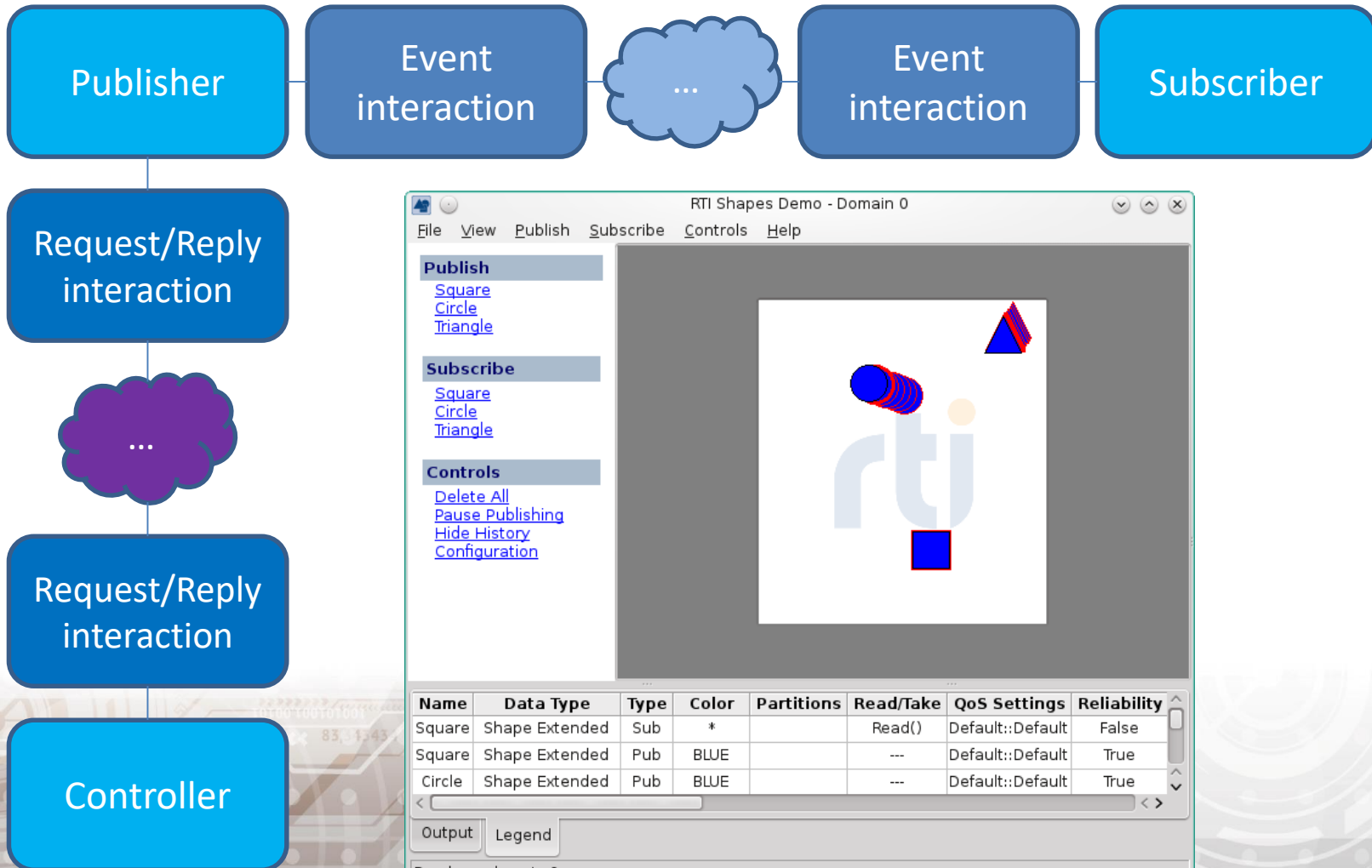


Shapes Example



Remedy IT

Your challenge - our solution



RTI Shapes Demo - Domain 0

File View Publish Subscribe Controls Help

Publish

- [Square](#)
- [Circle](#)
- [Triangle](#)

Subscribe

- [Square](#)
- [Circle](#)
- [Triangle](#)

Controls

- [Delete All](#)
- [Pause Publishing](#)
- [Hide History](#)
- [Configuration](#)

Name	Data Type	Type	Color	Partitions	Read/Take	QoS Settings	Reliability
Square	Shape Extended	Sub	*		Read()	Default::Default	False
Square	Shape Extended	Pub	BLUE		---	Default::Default	True
Circle	Shape Extended	Pub	BLUE		---	Default::Default	True

Output Legend

Ready on domain 0



Remedy IT

Generated ShapeType Class



Your challenge - our solution

```
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 shapessize);
    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 shapel = shape;
ShapeType shape2 (shapel);
```



Remedy IT

Component Executor Class



Your challenge - our solution

```
/// Component Executor Implementation Class : Publisher_comp_exec_i
class Publisher_comp_exec_i final
  : public virtual IDL::traits<CCM_Publisher_comp>::base_type
{
public:
  /// Constructor
  Publisher_comp_exec_i ();
  //@@{__RIDL_REGEN_MARKER__} - END :
          Shapes_Publisher_comp_Impl::Publisher_comp_exec_i[ctor]

  /// Destructor
  virtual ~Publisher_comp_exec_i ();

  /** @name Component port operations. */
  //@{
  /// Factory method and getter for the control facet
  /// @return existing instance of facet if one exists, else creates it
  virtual 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;
}
```



Conclusion



- DDS fits perfect into a component based approach
- DDSX11 abstracts vendor differences and improves portability of user code
- Fully automated testing is possible but takes time to implement
- 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

tel.: +31(0)88 053 0000

e-mail: sales@remedy.nl

website: www.remedy.nl

Twitter: [@RemedyIT](https://twitter.com/@RemedyIT)

Slideshare: [RemedyIT](https://www.slideshare.net/RemedyIT)