

the Metahint Impl Generator. In **FIG. 15**, the Attribute Handler of the Metahint Impl Generator creates Attribute Implementations of the Metahint Implementation. In addition to the Attribute Implementations, the Metahint Implementations includes the Metahint Descriptor it implements. The Name Attr(tribute) Impl(ementation) of the Attribute Implementations creates a Name of the Attribute Instances of the Hint Descriptor. The Value Attr(tribute) Impl(ementation) of the Attribute Implementations creates a Value of the Attribute Instances of the Hint Descriptor. Any other attribute implementations (Other Attr Impl) of the Attribute Implementations create attributes appropriately (Other Attribute<n>) in the Hint Descriptor. In addition to the Attribute Instances, the Hint Descriptor includes the Metahint Implementation of which Hint Descriptor is an instance.

[0472] The Hint Descriptor is read by the Hint Impl(ementation) Generator, which creates a Hint Implementation. An Attribute Handler of the Hint Impl Generator creates a Name, Value and other attributes (Other Attributes<n>) of the hint implementation. The Hint Implementation also includes the Hint Descriptor of which the Hint Implementation is an implementation. The Name, Value and Other Attributes of the Hint Implementation create the Name, Value and Other Attributes<n>, respectively of the Hint Instance. The Hint Instance also includes a pointer to the Hint Implementation that created it of which it an instance.

[0473] In **FIG. 14**, a Tagged Value description is read by a Tagged Value Handler to create a Hint Instance. The Tagged Value handler uses the appropriate Hint Implementation to create the Hint Instance then maps the Name and Value from the Tagged Value to the Name Attribute Instance and Value Attribute Instance held by the Hint Instance.

[0474] Although not shown in every figure of **FIGS. 5-34**, every descriptor is defined by a Meta<Element> descriptor just as pictured in the Metahint Descriptor of **FIG. 14**.

[0475] As shown in **FIG. 14**, a user defines a Role Descriptor including a Description, Hint Names and Role Names. As indicated by connector **1** of **FIGS. 14 and 15**, the Role Descriptor is ready by a Role Impl Generator of **FIG. 15**. The Role Descriptor is saved to the Metamodel Repository of the Component Integration Engine.

[0476] A Role Implementation of the Component Integration Engine includes a pointer to the Role Descriptor for which the Role Implementation is an implementation and looks up roles and looks up hints from the Metamodel Repository. The Metamodel Repository is used to lookup the Hint Implementation to create Hint Instances for each Hint Name held by the Role Implementation. The Metamodel Repository is also used to lookup the Role Implementation to create Role Instances for each Role Name held by the Role Implementation.

[0477] Also as shown in **FIG. 14**, a Stereotype is read by a Role Descriptor Handler and creates a Role Descriptor. A Name of the Stereotype is handled by a Name Handler of the Role Descriptor Handler to set the value of the Name Attribute Instance on the Role Descriptor. Tagged Values of the Stereotype are handled by a Tagged Value Handler of the Role Descriptor Handler to create the Hint Names of the Role Descriptor.

[0478] In **FIGS. 16 and 17** a DataType is read by a DataType Descriptor Handler to create a DataType Descrip-

tor. The data type descriptor is read by a DataType Impl(ementation) Generator, as indicated by connector **J**, to create a DataType Implementation. The DataType Implementation can then be used to create DataType Instances. A Name, Documentation, a Parent DataType, Operations, Constraints and Tagged Values of the DataType are handled by a Name Handler, a Documentation Handler, a Parent DataType Handler, an Operations Handler, a Constraints Handler and a Tagged Values Handler, respectively, of the DataType Description Handler. The Name Handler of the DataType Descriptor Handler handles the Name of the DataType Descriptor. The Documentation Handler of the DataType Descriptor Handler handles the Description of the DataType Descriptor. The Parent DataType Handler of the DataType Descriptor Handler handles the Parent Type Name of the DataType Descriptor. The Operations Handler of the DataType Descriptor Handler creates the Operation Descriptors of the DataType Descriptor. The Tagged Values Handler of the DataType Descriptor Handler creates the Hint Instances of the DataType Descriptor.

[0479] The Operations Descriptor is read by the Operations Handler of the DataType Impl Generator as indicated by connector **K**. The Constraint Descriptors are read by the Constraint Handler of the DataType Impl Generator, as indicated by connector **L**. The Initial Value Configuration of the DataType Descriptor is read by the Initial Value Handler, as indicated by connector **M**. The DataType Descriptor also includes Role Instances added by a user to further capture details about the datatype which are not adequately captured anywhere else.

[0480] The Operation Handler of the DataType Impl Generator creates the Operation Implementations of the DataType Implementation. The Constraint Handler of the DataType Impl Generator creates the Constraint Implementations of the DataType Implementation. The Initial Value Handler of the DataType Impl Generator creates the Initial Value Instance of the DataType Implementation. The DataType Implementation also includes a pointer to the DataType Descriptor of which the DataType Implementation is an implementation.

[0481] The Operation Implementations of the DataType Implementation creates Operation Instances of the DataType Instance. The Constraint Implementations of the DataType Implementation create Constraint Instances of the DataType Instance. The Initial Value Instance creates Value(s) of the DataType Instance. The DataType Instance also includes a pointer to the DataType Implementation of which the DataType Instance is an instance.

[0482] In **FIGS. 18 and 19** a Constraint is read by a Constraint Descriptor Handler to create a Constraint Descriptor. The constraint descriptor is read by the Constraint Impl(ementation) Generator, as indicated by connector **N** to create a Constraint Implementation. The Constraint Implementation can then be used to create Constraint Instances. A Name, Documentation, an OCL Body, and Tagged Values of the Constraint description are handled by the Name Handler, Documentation Handler, OCL Handler, Tagged Value Handler, and Language Handler, respectively, of the Constraint Descriptor Handler.

[0483] The Name Handler, Documentation Handler and OCL Handler of the Constraint Descriptor Handler map to the Name, Description and OCL Rule Descriptor, respec-