Modeling Support for Confidentiality and Integrity of Object Flows in Activity Models

Bernhard Hoisl[1,2] and Mark Strembeck[1,2]

[1] Institute for Information Systems and New Media,
Vienna University of Economics and Business (WU Vienna), Austria
[2] Secure Business Austria (SBA) Research Center, Austria
{bernhard.hoisl,mark.strembeck}@wu.ac.at

Summary. While the demand for an integrated modeling support of business processes and corresponding security properties has been repeatedly identified in research and practice, standard modeling languages do not provide native language constructs to model process-related security properties. In this paper, we are especially concerned with confidentiality and integrity of object flows. In particular, we present an UML extension called SecureObjectFlows to model confidentiality and integrity of object flows in activity models. Moreover, we discuss the semantics of secure object flows with respect to control nodes and provide a formal definition of the corresponding semantics via the Object Constraint Language (OCL).

Key words: Activity Models, Modeling Security Properties, Process Modeling, UML

1 Introduction

IT systems must comply with certain laws and regulations, such as the Basel II Accord, the International Financial Reporting Standards (IFRS), or the Sarbanes-Oxley Act (SOX). For example, adequate support for the definition and enforcement of process-related security policies is one important part of SOX compliance (see, e.g., [1, 2]). Moreover, corresponding compliance requirements also arise from security recommendations and standards such as the NIST security handbook [3], the NIST recommended security controls [4], or the ISO 27000 standard family (formerly ISO 17799).

While the demand for an integrated modeling support of business processes and corresponding security properties has been repeatedly identified (see, e.g., [5, 6]), different types of problems arise when modeling process-related security properties. First, contemporary modeling languages such as BPMN (Business Process Model and Notation, [7]) or UML activity models (Unified Modeling Language, [8]) do not provide native language constructs to model secure object flows. A second problem is that the language used for process modeling is often different from (or not integrated with) the system modeling language that is used to specify the corresponding software system. This, again, may result in problems because different modeling languages provide different language abstractions that cannot easily be mapped to each other. In particular, such semantic gaps may involve significant efforts when conceptual models from different languages need to be integrated and mapped to a software platform (see, e.g., [9, 10]).

However, a complete and correct mapping of process definitions and related security properties to the corresponding software system is essential in order to assure consistency between the modeling-level specifications on the one hand, and the software system that actually manages corresponding process instances and enforces the respective security properties on the other.

In this paper, we are concerned with the modeling of secure object flows in process models – in particular UML activity diagrams. UML is a de facto standard for software systems modeling. It provides a family of integrated modeling languages for the specification of the different aspects and perspectives that are relevant for a software system. Therefore, to demonstrate our approach, we chose to define an extension to the UML metamodel that allows to specify confidentiality and integrity properties of object flows in activity models. Activity models have a token semantics, and object tokens are passed along object flow edges (for details see [8]). Thus, to ensure the consistency of the corresponding activity models, it is especially important to thoroughly specify the semantics of secure object flows with respect to control nodes (such as fork, join, decision, and merge nodes). Therefore, we use the Object Constraint Language (OCL, [11]) to formally define the semantics of our extension. Corresponding software tools can enforce the OCL constraints on the modeling-level as well as in runtime models. Thereby, we can ensure the consistency of the extended activity models with the respective constraints.

The remainder of this paper is structured as follows. In Section 2 we present our UML extension for secure object flows in activitiy models. Subsequently, Section 3 discusses the semantics of secure object flows, with a special focus on the semantics arising from different types of control nodes. Next, Section 4 discusses related work, and Section 5 concludes the paper.[1]

2 UML Extension for Secure Object Flows

Thereby, *confidentiality* ensures that important/classified objects (such as business contracts, court records, or electronic patient records) which are used in a business process can only be read by designated subjects (see, e.g., [4, 12]). *Integrity* ensures that important objects are in their original/intended state, and enable the straightforward detection of accidental or malicious changes (see, e.g., [3, 13, 14]).

To provide modeling support for confidentiality and integrity properties of object flows, we define a new package *SecureObjectFlows* as an extension to

[1] We provide an extended version of this paper on our Web page. In the extended version we re-inserted the text that we had to cut from the paper due to the page restrictions for the proceedings version.

the UML metamodel (see Fig. 1). In particular, we introduce *SecureNode*, *SecurePin*, *SecureDateStoreNode*, and *SecureActivityParameterNode* as new modeling elements. A secure object flow is defined as an object flow between two or more of the above mentioned secure object nodes. The SecureNode element is defined as an abstract node, and the SecurePin, SecureDataStoreNode, and SecureActivityParameterNode represent specialized secure nodes. In particular these three nodes inherit the properties from their corresponding parent object nodes as well as the security related properties from SecureNode (see Fig. 1).

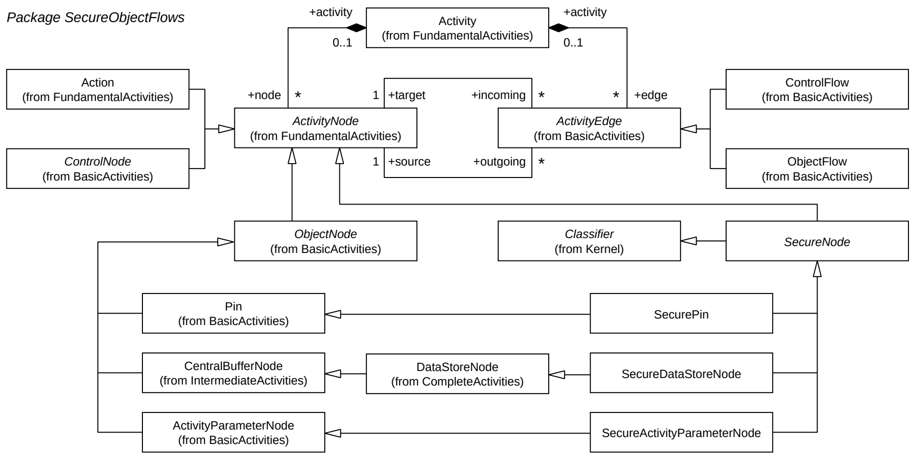


Fig. 1. UML metamodel extension for secure object flows.

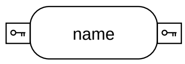
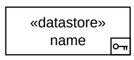
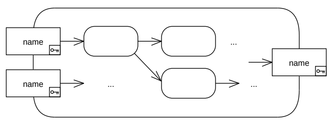
Below, we specify the attributes of the SecureNode elements defined via the metamodel extension. In addition, we use the OCL to formally specify the semantics of the SecureObjectFlows package. For the sake of readability, we decided to move the associated OCL constraints to Appendix A. However, these OCL constraints are a significant part of our UML extension, because they formally define the semantics of the new modeling elements. Therefore, each UML model that uses the SecureObjectFlows package must conform to these OCL constraints.

- *confidentialityAlgorithm : Classifier [0..1]*
  References a classifier that provides methods to ensure confidentiality properties of the object tokens that are sent or received by a SecureNode, e.g. a class implementing DES (Data Encryption Standard) or AES (Advanced Encryption Standard) functionalities.
- *confidentialityKeyLength : Integer [0..1]*
  Defines the key length of encryption method used, for example 256 bit.
- *confidentialityEnsured : Boolean [0..1]*
  This Attribute is derived from the attributes *confidentialityAlgorithm* and *confidentialityKeyLength*. It evaluates to true if a SecureNode supports confidentiality-related security properties (see OCL Constraint 1).

– *integrityAlgorithm : Classifier [0..1]*
  References a classifier that provides methods to ensure integrity properties of the object tokens that are sent or received by a SecureNode, e.g. a class implementing SHA-1 or SHA-384 (Secure Hash Algorithm) functionalities.
– *integrityEnsured : Boolean [0..1]*
  This attribute is derived from the attribute *integrityAlgorithm*. It evaluates to true if a SecureNode supports integrity-related security properties (see OCL Constraint 2).

With respect to the attributes defined above, we specify that a secure object node either supports confidentiality properties, or integrity properties, or both (see OCL Constraint 3). Table 1 shows the graphical elements for SecureNodes.

Table 1. Notation of elements for modeling secure objects.

Node Type	Notation	Explanation
SecurePin (attached to action)		A SecurePin attached to an action is shown as a UML Pin element that includes a key symbol.
SecureData-StoreNode		A SecureDataDtoreNode is shown as a UML DataStoreNode element with a key symbol in the lower right corner surrounded by a small rectangle.
Secure-Activity-Parameter-Node		A SecureActivityParameterNode is shown as a UML ActivityParameter-Node element with a key symbol in the lower right corner surrounded by a small rectangle.

3 Semantics of Secure Object Flows

The main element of an activity model is an activity. Actions define the tasks (steps) that are performed when executing the corresponding activity. Activity models have a token semantics, similar (but not equal) to petri nets (for details see [8]). In general, two different types of tokens can travel in an activity model. Control tokens are passed along control flow edges and object tokens are passed along object flow edges. This means, each type of token is exclusively passed along edges of the corresponding edge type.

A decision node chooses between outgoing flows and, therefore, has one incoming and multiple outgoing edges. Decision nodes do not duplicate tokens. Therefore, each token arriving at a decision node can travel along exactly one outgoing edge. A merge node consolidates multiple incoming flows and thus has multiple incoming and one outgoing edge. However, merge nodes do not synchronize concurrent flows nor do they join incoming tokens. Thus, each token arriving at a merge node is offered to the outgoing edge. Both, decision and merge nodes are represented by a diamond-shaped symbol respectively. A fork node splits a flow into multiple concurrent flows and thus has one incoming and multiple outgoing edges. Tokens arriving at a fork node are duplicated and passed along each edge that accepts the token. A join node synchronizes multiple flows and therefore has multiple incoming and one outgoing edge. A join node may join/combine incoming tokens (in contrast to merge nodes, see above). Both, fork and join nodes are represented via a thick line (for details see [8]).

To ensure the consistency of the corresponding activity models, it is especially important to thoroughly specify the semantics of secure object flows. Otherwise, a combination of ordinary object flows and secure object flows could result in inconsistencies. Therefore, Section 3.1 discusses the semantics of secure object nodes with respect to direct object flows, Section 3.2 discusses the semantics with respect to decision and merge nodes, and Section 3.3 with respect to fork and join nodes.

3.1 Semantics of Secure Object Nodes regarding Direct Object Flows

We use the term *direct object flow* to refer to an object flow that directly connects object nodes without intermediate control nodes. Fig. 2 shows three example configurations of direct object flows involving SecureNodes. All statements and OCL constraints referenced below refer to SecureNode and therefore apply for each subtype of SecureNode (see Fig. 1).

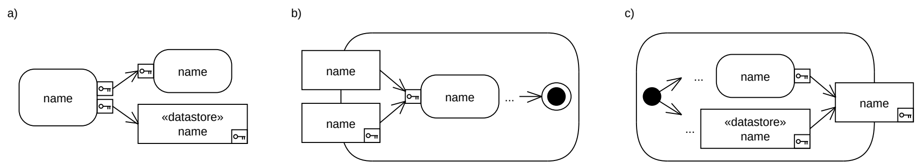


Fig. 2. Examples of direct object flows between secure nodes.

Fig. 2a shows a configuration where two SecurePins attached to an action serve as data sources for two other secure object nodes. To ensure a secure object flow, we define that if an object node receives an object token from a SecureNode, the target node must also be a SecureNode (see OCL Constraint 4). Otherwise, a secure object flow could have a SecureNode as its source and an ordinary object node as its target – which would result in an inconsistency because ordinary object nodes cannot ensure the confidentiality or integrity of object tokens.

Because each subtype of SecureNode does also inherit the properties of the corresponding ordinary UML object node (see Fig. 1), it can process ordinary object tokens as well as secure object tokens. Fig. 2b shows a configuration where an ordinary ActivityParameterNode and a SecureActivityParameterNode serve as source nodes for a SecurePin. In such a configuration, the target node must be a SecureNode (see OCL Constraint 4) and the target node (here a SecurePin) must support the same security properties as the corresponding secure source node (here a SecureActivityParameterNode). This requirement is formally specified via OCL Constraint 5.[2] This constraint guarantees that the security properties of object tokens sent by a certain source node can be checked and ensured by the corresponding target node(s).

Fig. 2c shows a configuration where a SecurePin and a SecureDataStoreNode serve as source nodes for a SecureActivityParameterNode. Thus, according to OCL Constraint 4, the target node must also be a SecureNode (here it is a SecureActivityParameterNode) and it must support all security properties that are supported by the respective source nodes (see OCL Constraint 5). Moreover, we define that all source nodes must provide compatible security properties (see OCL Constraint 6). Otherwise, the source nodes could use, for example, different cryptographic algorithms or different key lengths – which could again result in inconsistencies and in a violation of OCL Constraint 5.

3.2 Semantics of Secure Object Flows regarding Decision and Merge

Fig. 3 shows examples of the different configuration options of secure object flows that include decision or merge nodes.[3] Fig. 3a shows a configuration where a decision node has an incoming secure object flow and presents the corresponding object tokens to multiple outgoing edges. As the source of the incoming object flow is a SecureNode (here it is a SecurePin) both target nodes must also be secured (see OCL Constraint 7). Otherwise, a secure object flow could have a SecureNode as its source and an ordinary object node as its target – which would result in an inconsistency because ordinary object nodes cannot ensure confidentiality or integrity of object tokens. Furthermore, target nodes of a secure object flow must support the same security properties as the respective source node (see OCL Constraint 8). This constraint ensures that security properties cannot be lost when traversing a decision node and that the target node(s) are able to check and ensure the corresponding security properties.

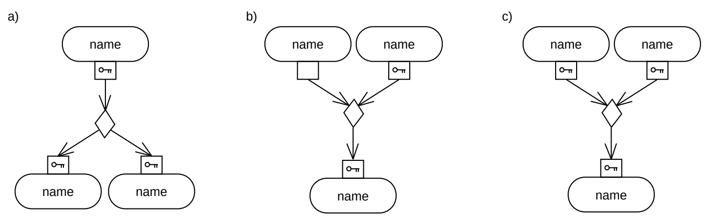


Fig. 3. Secure object flows with decision and merge nodes.

[2] Note that the OCL invariants from Appendix A complement each other.

[3] For the sake of simplicity, Fig. 3 as well as Fig. 4 show only two incoming/outgoing flows for the respective control nodes. However, the corresponding OCL constraints apply for an arbitrary number of incoming/outgoing edges, of course.

Fig. 3b shows a configuration where a merge node brings together alternate flows – one of which is a secure object flow. For such a configuration, we define that if a merge node receives at least one secure object flow, the target node of this merge node must also be a SecureNode (see OCL Constraint 9). This constraint guarantees that each secure object token passing a merge node can be checked and processed by the corresponding target node.

Fig. 3c shows a configuration where a merge node brings together alternate secure object flows. According to OCL Constraint 9 the target must be a SecureNode. Furthermore, we define that all source nodes must provide compatible security properties (see OCL Constraint 10). In addition, the target node must support all security properties of the respective source nodes (OCL Constraint 11). Otherwise, incompatibilities could emerge if the security properties supported by the source nodes are different from the security properties supported by the target node.

3.3 Semantics of Secure Object Flows regarding Fork and Join

Fig. 4 shows examples of the different configuration options of secure object flows that include fork or join nodes. Fig. 4a shows a configuration where a fork node splits a secure object flow into multiple concurrent flows. Because the tokens arriving at a fork node are duplicated, all target nodes must be SecureNodes (see OCL Constraint 12). Furthermore, the target nodes must support the same security properties as the corresponding source node (see OCL Constraint 13). This constraint ensures that security properties cannot be lost when traversing a fork node and that the target node(s) are able to check and ensure the corresponding security properties.

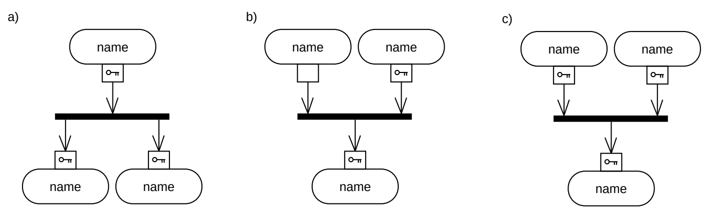


Fig. 4. Secure object flows with fork and join nodes.

Fig. 4b shows a configuration where a join node synchronizes multiple object flows – one of which is a secure object flow. Because in this case the join node receives secure as well as ordinary object tokens, we define that the tokens cannot be combined (see OCL Constraint 14). Moreover, we define that if a join node receives at least one secure object flow, then the target node of this join node must also be a SecureNode (see OCL Constraint 15). This constraint guarantees that each secure object token passing a join node can be checked and processed by the corresponding target node.

Fig. 4c shows a configuration where a join node synchronizes multiple secure object flows. As defined in OCL Constraint 15 the target must be a secure node. Furthermore, all source nodes must support compatible security properties (OCL Constraint 16). In addition, the target node must support all security properties of the corresponding source nodes (see OCL Constraint 17). Otherwise, inconsistencies could emerge if the security properties supported by the source nodes are different from the security properties supported by the target node.

4 Related Work

Several approaches exist to integrate process models with specific types of security policies and/or constraints on different abstraction levels. Jensen and Feja present an approach to specify three types of security properties (access control, confidentiality, and integrity) in Event-driven Process Chains [15]. Another related approach is UMLsec [16]. In essence, it provides a UML profile for the definition and analysis of security properties for software systems. For example, UMLsec is used to define and verify cryptographic protocols. However, UMLsec aims at a lower abstraction layer than our SecureObjectFlows extension. Therefore, UMLsec is well-suited to be combined with our approach. SecureObjectFlows would then be used to model business processes and process-level security properties, while UMLsec would be used to specify the fine-grained system-level procedures for encryption and integrity checking in a particular software system. Furthermore, Basin et al. [17] present a sophisticated approach called model-driven security. They demonstrate their approach with an UML profile for RBAC (Role-Based Access Control) called SecureUML. Here, the focus is on integrating security aspects with a model-driven development approach rather than modeling of business processes and process-related security properties. In fact, the model-driven security approach of SecureUML and our SecureObjectFlows package are well-suited to be combined in a complementary fashion.

5 Conclusion

A complete and correct mapping of process definitions and related security properties to the corresponding software system is essential in order to assure consistency between the modeling-level specifications on the one hand, and the software system that actually manages corresponding process instances and enforces the respective security properties on the other hand.

UML activity models provide a process modeling language that is tightly integrated with other model types from the UML family (such as class models, state machines, or interaction models). In this paper, we presented SecureObjectFlows as an integrated approach to model confidentiality and integrity properties of object flows in UML activity diagrams. The semantics of our extension are formally defined via the OCL. Corresponding software tools can enforce these invariants on the modeling-level as well as in runtime models. Thereby, we can ensure the consistency of our SecureObjectFlows models with the respective constraints. Moreover, our extension can be applied to supplement other UML-based approaches and can be integrated in UML-based software tools.

References

1. Damianides, M.: How does SOX change IT? Journal of Corporate Accounting & Finance **15**(6) (2004)
2. Mishra, S., Weistroffer, H.R.: A Framework for Integrating Sarbanes-Oxley Compliance into the Systems Development Process. Communications of the Association for Information Systems (CAIS) **20**(1) (2007)
3. National Institute of Standards and Technology: An Introduction to Computer Security: The NIST Handbook. Special Publication 800-12, available at: http://csrc.nist.gov/publications/nistpubs/800-12/handbook.pdf (1995)
4. National Institute of Standards and Technology: Recommended Security Controls for Federal Information Systems and Organizations. NIST Special Publication 800-53, Revision 3, available at: http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf (2009)
5. Botha, R.A., Eloff, J.H.P.: Separation of Duties for Access Control Enforcement in Workflow Environments. IBM Systems Journal **40**(3) (2001)
6. Wainer, J., Barthelmes, P., Kumar, A.: W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. International Journal of Cooperative Information Systems (IJCIS) **12**(4) (December 2003)
7. Object Management Group: Business Process Model and Notation (BPMN) - Version 2.0 - Beta 2. Available at: http://www.omg.org/spec/BPMN/2.0/Beta2/PDF (2010)
8. Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure - Version 2.3. Available at: http://www.omg.org/spec/UML/2.3/Superstructure/PDF/ (2010)
9. Axenath, B., Kindler, E., Rubin, V.: AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects. In Leymann, F., Reisig, W., Thatte, S.R., van der Aalst, W., eds.: The Role of Business Processes in Service Oriented Architectures. Number 06291 in Dagstuhl Seminar Proceedings (2006)
10. Zdun, U.: Patterns of Component and Language Integration. In: D. Manolescu, M. Voelter, J. Noble (editors): Pattern Languages of Program Design 5. (2006)
11. Object Management Group: Object Constraint Language - Version 2.2. Available at: http://www.omg.org/spec/OCL/2.2/PDF (2010)
12. Committee on National Security Systems: National Information Assurance (IA) - Glossary. Available at: http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf (2010)
13. National Security Agency: Information Assurance Technical Framework. Available at: http://handle.dtic.mil/100.2/ADA393328 (2000)
14. Sandhu, R.S.: On Five Definitions of Data Integrity. In: Proceedings of the IFIP WG11.3 Working Conference on Database Security VII. (1993)
15. Jensen, M., Feja, S.: A Security Modeling Approach for Web-Service-based Business Processes. In: 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, IEEE (April 2009) 340–347
16. Jürjens, J.: Secure Systems Development with UML. Springer Verlag (2005)
17. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: From UML Models to Access Control Infrastructures. ACM Transactions on Software Engineering and Methodology (TOSEM) **15**(1) (January 2006)

A Constraints for Secure Object Flows

This section provides the complete list of OCL-expressed derived values and invariants for the UML extension specified in Section 2.

OCL Constraint 1 *The confidentialityEnsured attribute of the SecureNode classifier is derived from the confidentialityAlgorithm and confidentialityKeyLength attributes and evaluates to true if confidentiality-related security properties are supported.*

```
context SecureNode::confidentialityEnsured : Boolean
derive: if confidentialityAlgorithm->notEmpty() and
           confidentialityKeyLength->notEmpty()
        then true else false
        endif
```

OCL Constraint 2 *The integrityEnsured attribute of the SecureNode classifier is derived from the integrityAlgorithm attribute. It evaluates to true if an integrity-related security property is supported.*

```
context SecureNode::integrityEnsured : Boolean
derive: if integrityAlgorithm->notEmpty()
        then true else false
        endif
```

OCL Constraint 3 *A secure object node must ensure either confidentiality, or integrity, or both.*

```
context SecureNode inv:
  self.confidentialityEnsured or
  self.integrityEnsured
```

OCL Constraint 4 *Any target of a secure object flow must also be a secure object node.*

```
context ObjectNode inv:
  if self.incoming->exists(i | i.source.oclIsKindOf(SecureNode))
  then self.oclIsKindOf(SecureNode) else self.oclIsKindOf(ObjectNode)
  endif
```

OCL Constraint 5 *The downstream secure object node must support at least all security properties supported by corresponding upstream secure object nodes.*

```
context SecureNode
inv: self.incoming->forAll(i |
      if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).confidentialityEnsured
      then self.confidentialityAlgorithm = i.source.oclAsType(SecureNode).confidentialityAlgorithm and
           self.confidentialityKeyLength = i.source.oclAsType(SecureNode).confidentialityKeyLength
       else true endif)
inv: self.incoming->forAll(i |
     if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).integrityEnsured
     then self.integrityAlgorithm = i.source.oclAsType(SecureNode).integrityAlgorithm
     else true endif)
```

OCL Constraint 6 *All secure object nodes having the same target node must support identical security properties.*

```
context SecureNode
inv: self.incoming->forAll(i1,i2 |
     if i1.source.oclIsKindOf(SecureNode) and i2.source.oclIsKindOf(SecureNode) and
        i1.source.oclAsType(SecureNode).confidentialityEnsured and i2.source.oclAsType(SecureNode).confidentialityEnsured
     then i1.source.oclAsType(SecureNode).confidentialityAlgorithm = i2.source.oclAsType(SecureNode).confidentialityAlgorithm and
          i1.source.oclAsType(SecureNode).confidentialityKeyLength = i2.source.oclAsType(SecureNode).confidentialityKeyLength
     else true endif)
inv: self.incoming->forAll(i1,i2 |
     if i1.source.oclIsKindOf(SecureNode) and i2.source.oclIsKindOf(SecureNode) and
        i1.source.oclAsType(SecureNode).integrityEnsured and i2.source.oclAsType(SecureNode).integrityEnsured
     then i1.source.oclAsType(SecureNode).integrityAlgorithm = i2.source.oclAsType(SecureNode).integrityAlgorithm
     else true endif)
```

OCL Constraint 7 *If a decision node has a secure source node, all target object nodes must also be secured.*

```
context DecisionNode inv:
  if self.incoming->exists(i | i.source.oclIsKindOf(SecureNode))
  then self.outgoing->forAll(o | o.target.oclIsKindOf(SecureNode))
  else true endif
```

OCL Constraint 8 *Target secure nodes of a decision node must support identical security properties as the corresponding source node.*

```
context DecisionNode
inv: self.incoming->forAll(i |
     if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).confidentialityEnsured
     then self.outgoing->forAll(o |
       o.target.oclAsType(SecureNode).confidentialityAlgorithm = i.source.oclAsType(SecureNode).confidentialityAlgorithm and
       o.target.oclAsType(SecureNode).confidentialityKeyLength = i.source.oclAsType(SecureNode).confidentialityKeyLength)
     else true endif)
inv: self.incoming->forAll(i |
     if.i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).integrityEnsured
     then self.outgoing->forAll(o |
       o.target.oclAsType(SecureNode).integrityAlgorithm = i.source.oclAsType(SecureNode).integrityAlgorithm)
     else true endif)
```

OCL Constraint 9 *If a merge node has at least one secure source node, the target must also be a secure node.*

```
context MergeNode inv:
  if self.incoming->exists(i | i.source.oclIsKindOf(SecureNode))
  then self.outgoing.target.oclIsKindOf(SecureNode)
  else true endif
```

OCL Constraint 10 *All secure source nodes that serve as input to a merge node must support the same security properties.*

```
context MergeNode
inv: self.incoming->forAll(i1,i2 |
     if i1.source.oclIsKindOf(SecureNode) and i2.source.oclIsKindOf(SecureNode) and
        i1.source.oclAsType(SecureNode).confidentialityEnsured and i2.source.oclAsType(SecureNode).confidentialityEnsured
     then i1.source.oclAsType(SecureNode).confidentialityAlgorithm = i2.source.oclAsType(SecureNode).confidentialityAlgorithm and
          i1.source.oclAsType(SecureNode).confidentialityKeyLength = i2.source.oclAsType(SecureNode).confidentialityKeyLength
     else true endif)
inv: self.incoming->forAll(i1,i2 |
     if i1.source.oclIsKindOf(SecureNode) and i2.source.oclIsKindOf(SecureNode) and
        i1.source.oclAsType(SecureNode).integrityEnsured and i2.source.oclAsType(SecureNode).integrityEnsured
     then i1.source.oclAsType(SecureNode).integrityAlgorithm = i2.source.oclAsType(SecureNode).integrityAlgorithm
     else true endif)
```

OCL Constraint 11 *The secure target node of a merge node must be capable of supporting all security properties of corresponding source nodes.*

```
context MergeNode
inv: self.incoming->forAll(i |
     if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).confidentialityEnsured
     then self.outgoing.target.oclAsType(SecureNode).confidentialityAlgorithm =
              i.source.oclAsType(SecureNode).confidentialityAlgorithm and
          self.outgoing.target.oclAsType(SecureNode).confidentialityKeyLength =
              i.source.oclAsType(SecureNode).confidentialityKeyLength
     else true endif)
inv: self.incoming->forAll(i |
     if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).integrityEnsured
     then self.outgoing.target.oclAsType(SecureNode).integrityAlgorithm = i.source.oclAsType(SecureNode).integrityAlgorithm
     else true endif)
```

OCL Constraint 12 *If a fork node has a secure source node, all target nodes must also be secured.*

```
context ForkNode inv:
  if self.incoming.source.oclIsKindOf(SecureNode)
  then self.outgoing->forAll(o | o.target.oclIsKindOf(SecureNode))
  else true endif
```

OCL Constraint 13 *Secure target nodes of a fork node must support the same security properties as the corresponding source node.*

```
context ForkNode
inv: if self.incoming.source.oclIsKindOf(SecureNode) and self.incoming.source.oclAsType(SecureNode).confidentialityEnsured
     then self.outgoing->forAll(o |
       o.target.oclAsType(SecureNode).confidentialityAlgorithm =
              self.incoming.source.oclAsType(SecureNode).confidentialityAlgorithm and
       o.target.oclAsType(SecureNode).confidentialityKeyLength =
              self.incoming.source.oclAsType(SecureNode).confidentialityKeyLength)
     else true endif
inv: if self.incoming.source.oclIsKindOf(SecureNode) and self.incoming.source.oclAsType(SecureNode).integrityEnsured
     then self.outgoing->forAll(o |
       o.target.oclAsType(SecureNode).integrityAlgorithm = self.incoming.source.oclAsType(SecureNode).integrityAlgorithm)
     else true endif
```

OCL Constraint 14 *If both, secure object nodes and ordinary object nodes are input to a join node, this join node must not combine the corresponding tokens.*

```
context JoinNode inv:
  self.incoming->forAll(i1,i2 |
    if i1.source.oclIsKindOf(SecureNode) and
       i2.source.oclIsKindOf(SecureNode) = false
    then self.isCombineDuplicate = false
    else true endif)
```

OCL Constraint 15 *If a join node has at least one secure source node, the corresponding target node must also be secured.*

```
context JoinNode inv:
  if self.incoming->exists(i | i.source.oclIsKindOf(SecureNode))
  then self.outgoing.target.oclIsKindOf(SecureNode)
  else true endif
```

OCL Constraint 16 *All secure source nodes of a join node must support the same security properties.*

```
context JoinNode
inv: self.incoming->forAll(i1,i2 |
     if i1.source.oclIsKindOf(SecureNode) and i2.source.oclIsKindOf(SecureNode) and
        i1.source.oclAsType(SecureNode).confidentialityEnsured and i2.source.oclAsType(SecureNode).confidentialityEnsured
     then i1.source.oclAsType(SecureNode).confidentialityAlgorithm = i2.source.oclAsType(SecureNode).confidentialityAlgorithm and
         i1.source.oclAsType(SecureNode).confidentialityKeyLength = i2.source.oclAsType(SecureNode).confidentialityKeyLength
     else true endif)
inv: self.incoming->forAll(i1,i2 |
     if i1.source.oclIsKindOf(SecureNode) and i2.source.oclIsKindOf(SecureNode) and
        i1.source.oclAsType(SecureNode).integrityEnsured and i2.source.oclAsType(SecureNode).integrityEnsured
     then i1.source.oclAsType(SecureNode).integrityAlgorithm = i2.source.oclAsType(SecureNode).integrityAlgorithm
     else true endif)
```

OCL Constraint 17 *The secure target node of a join node must be capable of supporting all security properties of corresponding source secure nodes.*

```
context JoinNode
inv: self.incoming->forAll(i |
     if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).confidentialityEnsured
     then self.outgoing.target.oclAsType(SecureNode).confidentialityAlgorithm =
              i.source.oclAsType(SecureNode).confidentialityAlgorithm and
          self.outgoing.target.oclAsType(SecureNode).confidentialityKeyLength =
              i.source.oclAsType(SecureNode).confidentialityKeyLength
     else true endif)
inv: self.incoming->forAll(i |
     if i.source.oclIsKindOf(SecureNode) and i.source.oclAsType(SecureNode).integrityEnsured
     then self.outgoing.target.oclAsType(SecureNode).integrityAlgorithm = i.source.oclAsType(SecureNode).integrityAlgorithm
     else true endif)
```