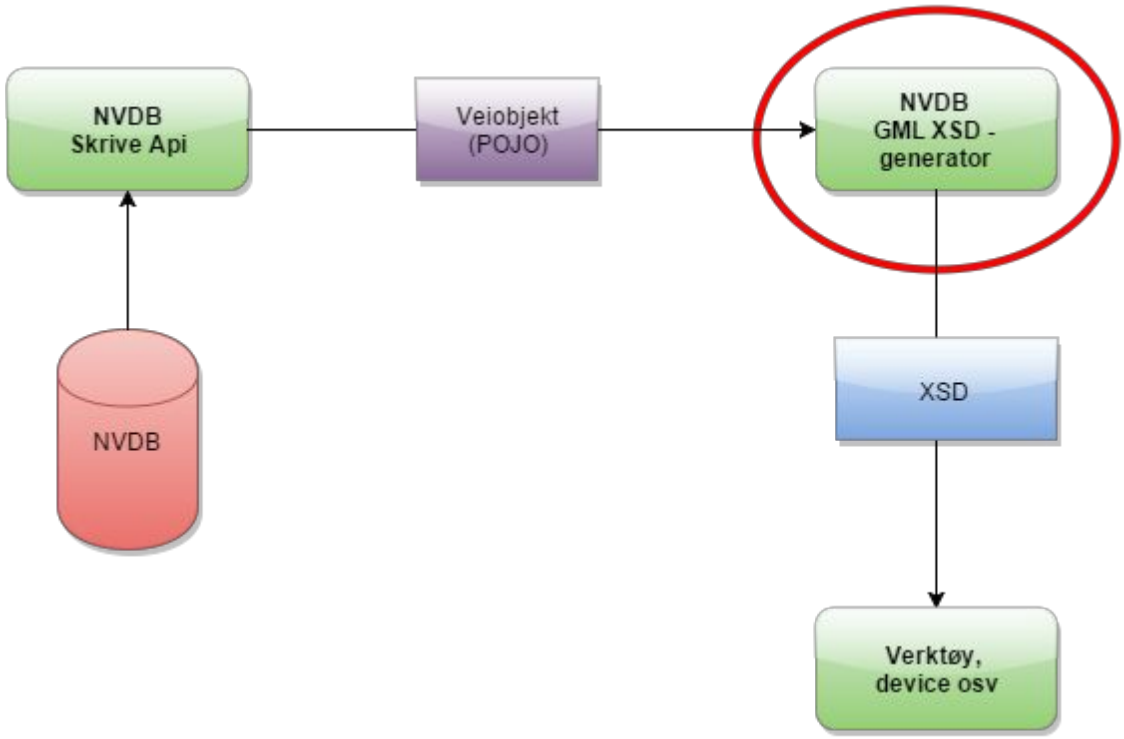
NVDB - GML XSD Generator

1. Mål og domene

Målet med prosjektet er å lage funksjonalitet for å kunne generere XSD (XML skjema), med mulighet for å arve fra GML, basert på veiobjekter lagret i datakatalogen (NVDB).
Systemet skal kunne mates med POJO (plain old java object) hentet ut fra Datakatalogen og gjøres om til XSD, som igjen kan brukes i verktøy, devicer o.l. brukt av statens vegvesen for å generere xml dokumenter med gyldig beskrivelse av veiobjekter.



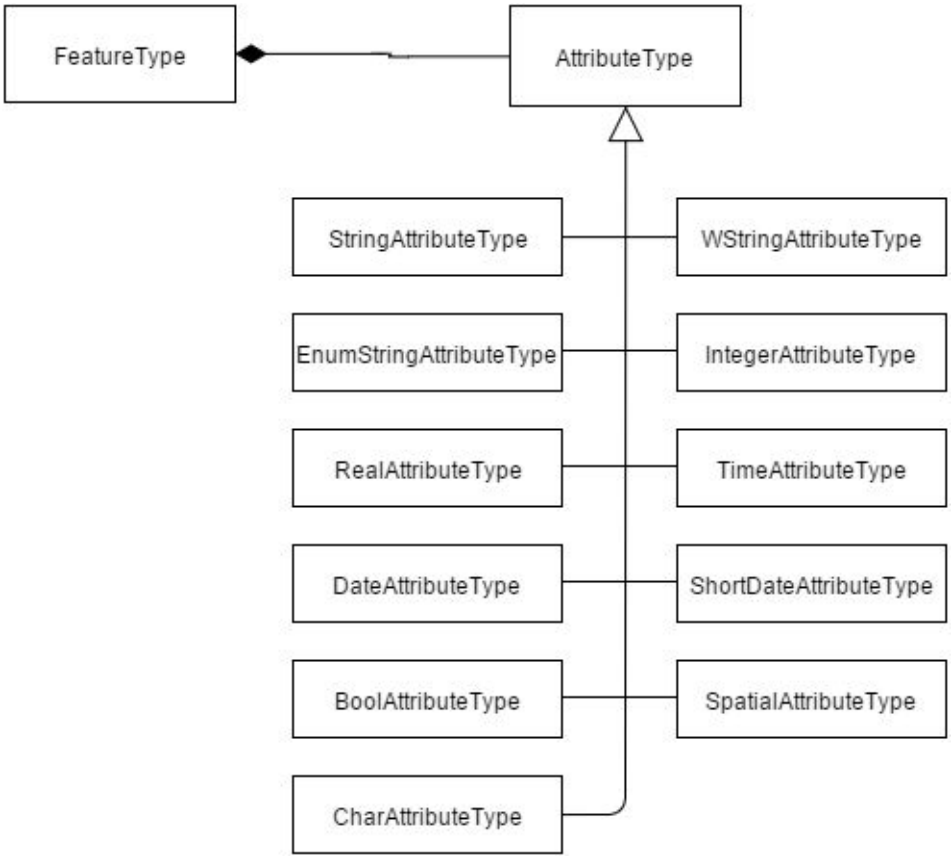
Figur 1: Domenemodell

2. Problembeskrivelse

Å generere en XSD (xml skjema) fra java klasser er ikke ny teknologi og kan automatiseres ved bruk av JAXB [https://jaxb.java.net/]. Dette er derimot kun brukbart hvis det er selve klasse definisjonen som beskriver hvordan XML skjemaet skal se ut. Med andre ord et felt i klassen blir beskrevet som et element i skjemaet.
Problemdomenet til NVDB er derimot ikke å generere XSD fra klassedefinisjoner, men heller definisjoner beskrevet av innholdet i feltene til klassen.

Her skal ikke en string atributt i java gjøres om til et string element i XSD, det er heller tildstedeværelsen av en instans av klassen StringAttributeType(se figur 2) i et featuretype objekt, som forteller at skjemaet skal inneholde et String element.

Det skal genereres skjema med elementer basert på innholdet i POJO og tilstedeværelsen av objekter av subklasser til attributetype, ikke basert på klassedefinisjonen til objektet.



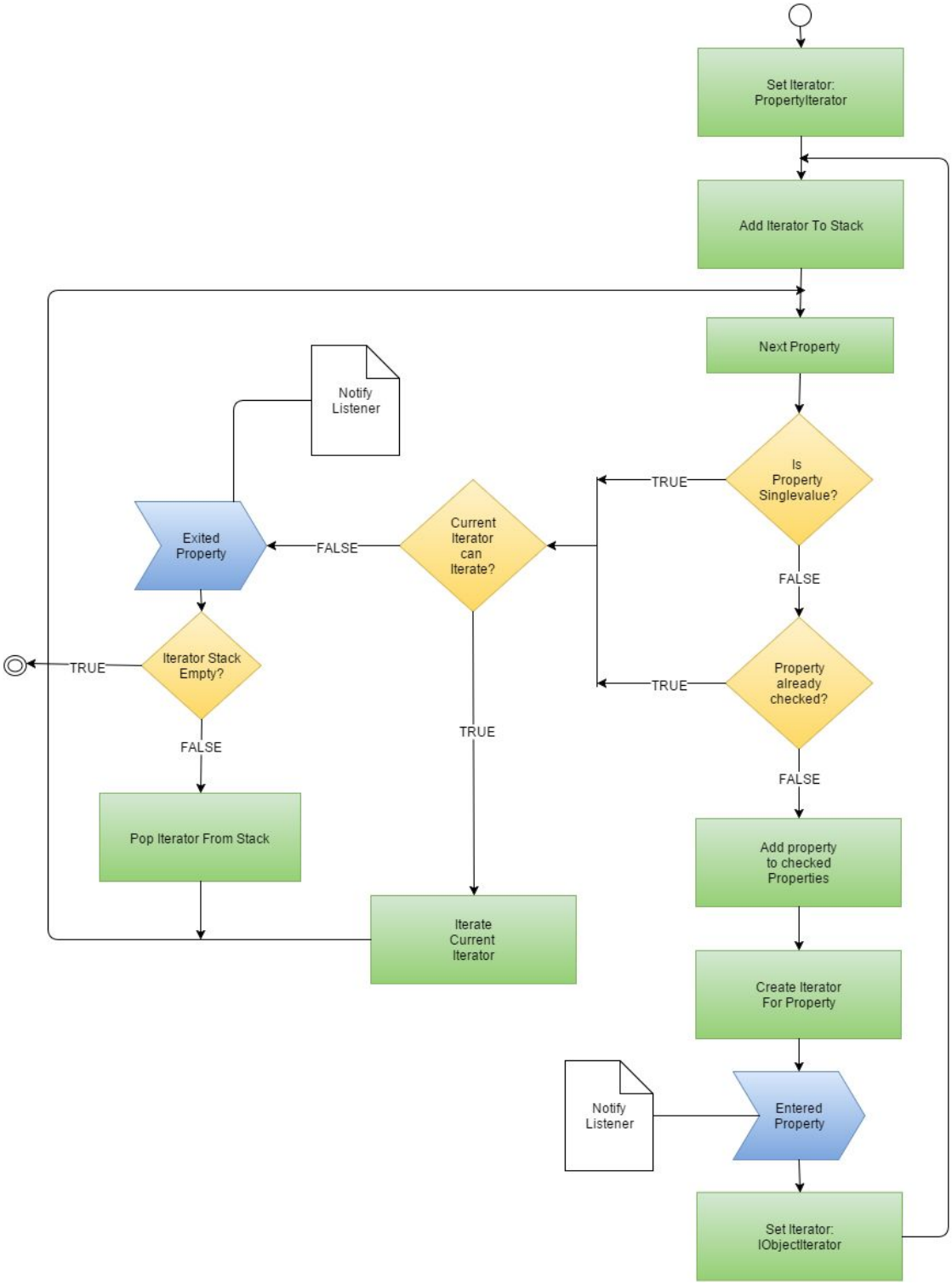
Figur 2: Featuretype og attributetype

3. Fremgangsmåte og resultat

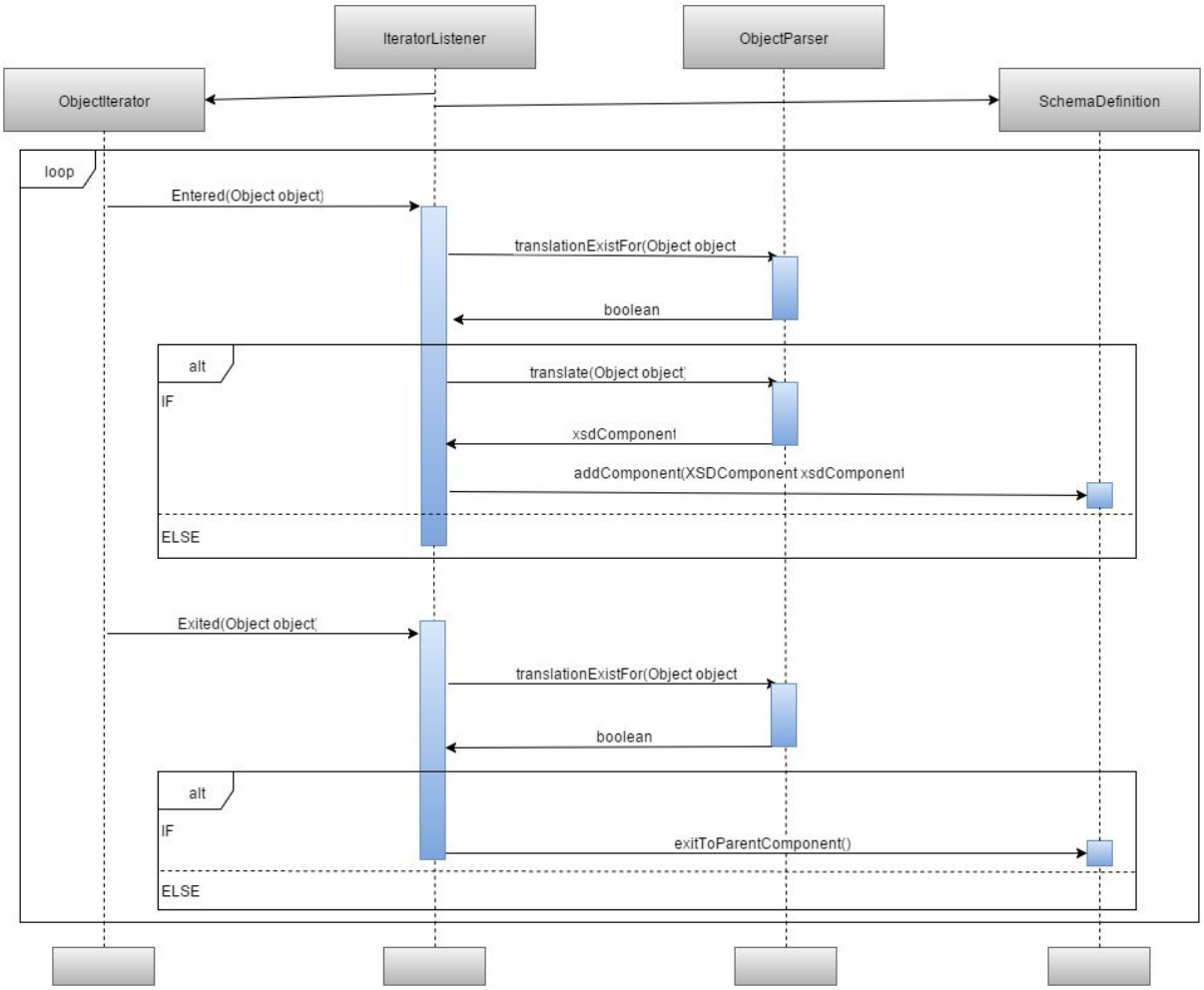
3.1 ObjectIterator

For å kunne beskrive et XML skjema fra objektgrafen til et java objekt, trengs det en måte å traversere objektgrafen på, for så å rapportere hvilken klasser grafen består av.
Målet var å lage funksjonalitet som kunne rapportere hver gang en ny klasse ble funnet i objektgrafen, og rapportere hierarkiet til grafen.
Dette blir oppnådd vha. “Iterator” modulen(for flytdiagram se figur 3).
Iterator modulen tar inn en POJO, og oppretter en iterator for hver atributt den finner i objektgrafen som ikke er en primær datatype. Finner iterator en instans av en klasse bruker den en PropertyIterator, finner iteratoren et set, array, map eller lignende, velges en passende iterator. Hver gang den må benytte seg av en ny iterator legges nåverende iterator

på en “stack”, og den nye iteratoren blir satt til nåværende iterator. Nåværende iterator vil alltid iterere til det enten ikke er fler elementer igjen i datastrukturen iteratoren er lagd for, eller til den selv blir plassert på stacken og må vike for en ny iterator. Når det ikke er fler elementer igjen for en gitt iterator “poppes” denne av stacken og underliggende iterator tar så over til denne også blir tomlementer. Dette foregår helt til alle properties på alle nivå i objektgrafen er blitt analysert. Hver gang *ObjectIterator* finner et objekt vil den si i fra til en IteratorListener, og sende med objektet den har funnet(se figur 4 for sekvensdiagram).



Figur 3: Flytdiagram “Iterator” modul



Figur 4: Sekvensdiagram for ObjectIterator, IteratorListener og ObjectParser.

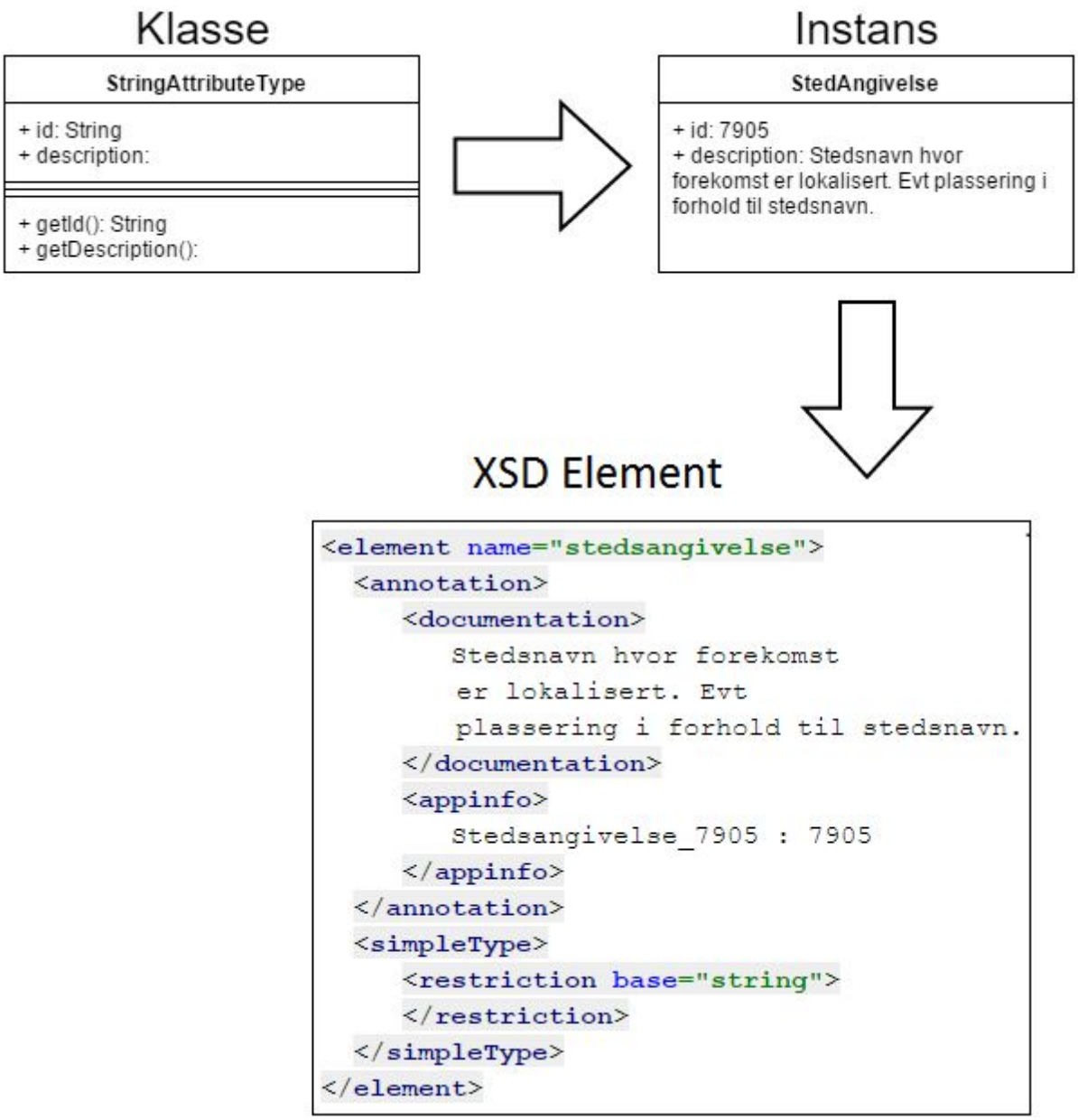
3.2 IteratorListener

IteratorListener lytter etter kall fra *ObjectIterator*.
Hver gang *IteratorListener* mottar et "entered" kall, spør *IteratorListener ObjectParser* om det er registrert en oversettelse av objektet. I tilfelle *ObjectParser* har en registrert oversettelse av klassen, kalles metoden *translate(Object object).* Translate metoden returnerer et XSDComponent objekt, som deretter blir registrert hos *SchemaDefinition*.
Hver gang *IteratorListener* mottar et "exited" kall spør også *IteratorListener ObjectParser* om det er registrert en oversettelse av objektet. I tilfelle det er registrert en oversettelse, kalles metoden *exitToParentComponent()* på *SchemaDefinition*.

3.3 ObjectParser

Det er i implementasjonen av Interfacet ObjectParser, domenets regler blir satt. Det er her objektene funnet av *ObjectIterator*, til slutt blir oversatt til XSDComponent. Klasser som ikke er definert i denne implementasjonen blir ignorert av systemet og vil ikke bli tatt med i skjema definisjonen.

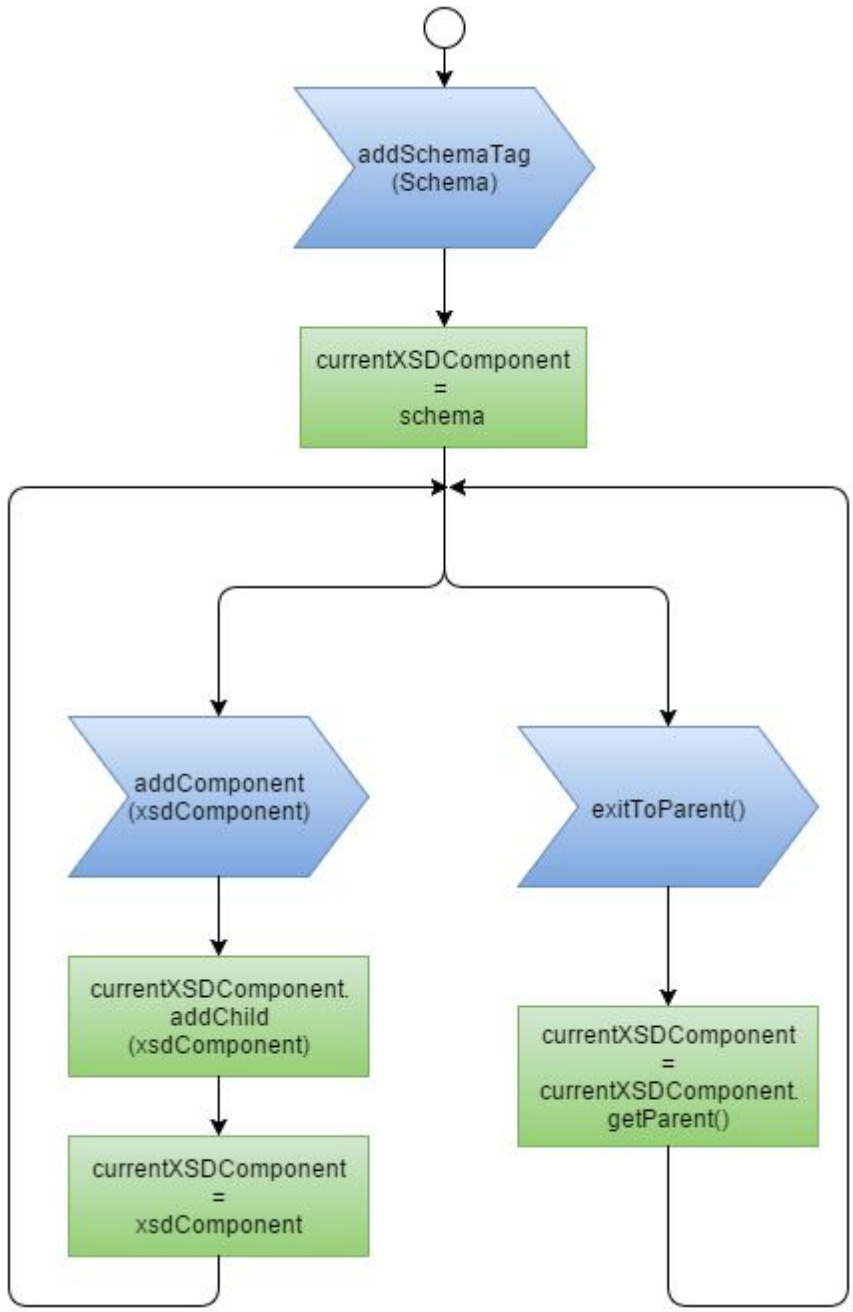
Utifra ønskelig struktur på XSD skjemaet kan man i implementasjonen av ObjectParser beskrive hvilke klasser mappes til hva slags XSD elementer (se figur 5 for simplifisert modell av forhold mellom klasse, instans og XSD element). Her er id atributten til klassen inkapsulert i et appinfo element, description atributten lagt i et documentation element og klassen selv er blitt brukt til å definere restriksjonen til elementets simpletype.



Figur 5: Simplifisert eksempel på forhold mellom klassen StringAttributetype, en instans av klassen og et generert xsd element.

3.4 SchemaDefinition

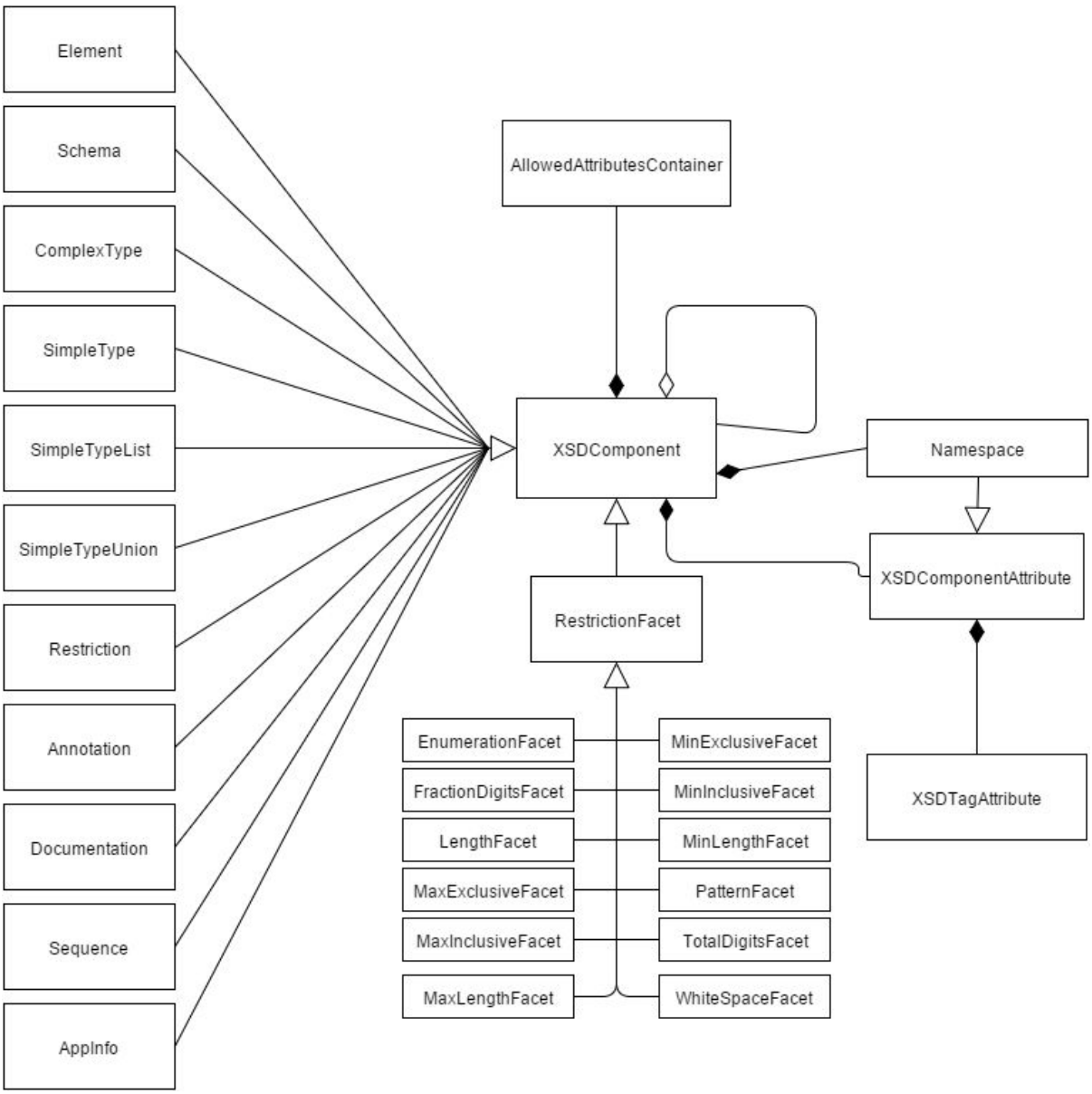
Denne klassen mottar *XSDComponent* objekter fra *IteratorListener* etter disse har blitt generert av *ObjectParser*. *SchemaDefinition* bruker et Composite Pattern for å opprette en trestruktur av *XSDComponenter*. Der rot noden er et *Schema XSDComponent* objekt, og alle kommende *XSDComponenter* opprettet av *ObjectParser* blir plassert på riktig plass i hierarkiet i henhold til objektgrafen, som barn av allerde registrerte *XSDComponenter*(se figur 6 for flytdiagram).



Figur 6: Flytdiagram for SchemaDefinition

3.5 XSDComponent

XSDComponent er representasjonen for et XSD element i systemet. *XSDComponent* er en abstrakt klasse med en subklasse for hver enkelt lovlige XSD element. *XSDComponent* benytter seg av *Composite Pattern*[https://en.wikipedia.org/wiki/Composite_pattern] der hvert element kan ha elementer av sin egen type, som "barne elementer".
Hver subklasse av *XSDComponent* har en instans av klassen *AllowedAttributesContainer*, som inneholder predefinerte, lovlige atributter for subklassen.
XSDComponent inneholder en eller flere instanser av *XSDComponentAttribute* klassen. Denne klassen er brukt som "container" for å holde på en *TagAttribute* (feks. *name* i figur 5. med verdien "*stedangivelse*").
Namespace atributten blir brukt for å kunne definere hvilket namespace komponenten hører til, og i tilfelle den hører til et namespace, legges prefixen til både komponenttaggen, men også verdien.



Figur 7: XSDComponent oppbygning