

## OPC UA vs. RESTful grensesnitt

Programvare-grensesnitt (software interface) er et område som kan koste produsenter av utstyr og programvare mye tid og penger. Her finnes et tilsynelatende virvar av ofte inkompatible standarder; API programmeringsgrensesnitt (Application Programming Interfaces), IO-kommunikasjon (feltbuss/nett) og webtjenester (Web services) – vi skal her se nærmere på HTTP med RESTful grensesnitt.

Med digitalisering, IoT (Internet of Things) og Industri 4.0 har det blitt stadig mer populært med webtjenester som grensesnitt mellom applikasjoner. Dette blir da en type API pakket inn i HTTP, som jo er basis for det meste vi foretar oss på Internett hver dag.

HTTP gir grunn-funksjonaliteten for å kommunisere data mellom to Internett-systemer, uten å vite noe om hva som sendes og mottas.

RESTful (eller bare REST) er en populær arkitekturstil som bruker HTTP-metoder (GET, PUT, POST, DELETE, PATCH). Den er tilstandsløs (stateless) og fullstendig generisk, laget for høy ytelse og pålitelighet. Man kan aksessere all informasjon på Internett som er representert som XML eller JSON-objekter.

En annen måte å standardisere programvare-grensesnitt er OPC UA. Den har åpnere transport – valg mellom binær TCP eller webtjeneste type HTTP/SOAP, bedre sikkerhet og en mer komplett informasjonsmodell enn den originale OPC (OPC Classic). UA er svært fleksibel og kan tilpasses når man vil kommunisere og knytte data mellom ulike typer styresystemer, overvåkingsenheter og sensorer som på den ene siden samler sanntidsdata fra den virkelige verden, og på den andre siden gjerne samhandle med bedriftssystemer – MES, entrepris ressursplanlegging (ERP), lagersystemer etc.

OPC UA er en åpen arkitektur standard-metode for kommunikasjon mellom applikasjoner, kontrollere, sensorer og andre enheter. Data i en OPC UA-enhet kan «oppdages». En klientenhet kan bla gjennom en annen OPC UA-enhet for å finne ut hvilke data som er tilgjengelige, navn, format og egenskaper (metadata egenskaper). Når man har funnet hvilke eksterne data man vil benytte i en applikasjon, kan disse leses, skrives eller planlegges for overføring med en bestemt frekvens.

Med OPC UA kan man ha skalerbare plattformer, ulike sikkerhetsmodeller, flere transportlag og sofistikerte informasjonsmodeller. Den minste dedikerte PLS kan fritt samhandle med komplekse, avanserte serverapplikasjoner. UA kan kommunisere alt fra enkel nedetidsstatus til enorme mengder svært kompleks informasjon for et helt anlegg eller fabrikk.

Informasjonsmodeller er selve basisen i OPC UA. Det er ikke noe annet enn en logisk representasjon av en fysisk prosess. En OPC UA informasjonsmodell kan representere noe så lite som en skrue, en komponent i en prosess, som en pumpe, eller noe komplekst og stort som en hel fyllemaskin. Informasjonsmodellen er ganske enkelt en struktur som definerer komponenten, uten all informasjon om hvordan prosessvariabler eller metadata innenfor den strukturen er gjort tilgjengelig.

Et vanlig spørsmål: "Hva er fordelene med OPC UA fremfor et RESTfult grensesnitt?" Ulempen med RESTful er at data vanligvis overføres som XML- og JSON-filer uten typedata og metadata. Det er ingen standardisert mekanisme for en klient til å tolke, skaffe dataoppsett eller beskrivelse av disse filene. Man får heller ikke tilgang til vanlige tjenester eller kommandoer (Start pumpe, hent pall, utfør oppskrift ... osv.) eller planlegge datafiloverføringer. RESTful er enkelt og rett frem, men på bekostning av funksjonalitet.

I motsetning til RESTful, har OPC UA mer robuste transport og koding av data, sikkerhetsmekanismer, tjenester og mulighet for å lage informasjonsmodeller. OPC UA gir en mer komplett enhetsdatamodell og dataverdier som, i motsetning til ASCII-filer overført av RESTful-arkitekturer, beholder sin opprinnelige datatype, presisjon og nøyaktighet.

For profesjonelle industriapplikasjoner er derfor OPC UA det riktige valget – med sine overlegne datamodelleringssegenskaper, presise datarepresentasjoner, en omfattende tjenesteinfrastruktur og fleksibilitet på lang sikt.