





About NXP Semiconductors

- Net Revenue: \$4.82 billion (2013)
- Established: 2006 (former division of Philips)
- 55+ years of experience in semiconductors
- Headquarters: Eindhoven, The Netherlands



Secure Connections for a Smarter World



- Businesses
 - High Performance Mixed Signal: Automotive, Identification, Infrastructure & Industrial, Portable & Computing
 - Standard Products
- Customers & distribution partners
 - 10 largest OEM customers: Apple, Bosch, Continental, Delphi, Gemalto, Giesecke & Devrient, Huawei, Nokia, Siemens Network, Samsung and ZTE
 - 3 largest distribution partners: Arrow, Avnet and WPG



About Semaku

- Established: 2013
- Headquarters: Eindhoven, The Netherlands
- Consultancy services and software products
- Areas of expertise
 - Product Information Management
 - Linked Data
 - Content Strategy
- http://semaku.com
- info@semaku.com







Linked Data at NXP So far, so good...

- Canonical data source for marketing master data
 - Provide the linking data
 - Unambiguous identifiers
- Using stored SPARQL SELECT queries to expose REST APIs
 - Results in tabular XML, JSON or CSV/TSV format
 - Easy to manage queries
 - Extremely quick to set up new APIs (15 30 mins)
- Able to answer previously unanswerable questions
- Minimal investment compared to traditional BW/BI projects



But we want more

Faster, faster, more, more...
 – Daily RDF dumps not frequent enough

- Do not contain all the data

Phase out legacy systems
 – Reduce maintenance effort/costs

Manage data natively as RDF





Adding a new data source

PRODUCT LIFECYCLE MANAGEMENT



What is product lifecycle management

- "product lifecycle management (PLM) is the process of managing the entire lifecycle of a product ... PLM integrates people, data, processes and business systems and provides a product information backbone for companies and their extended enterprise." – Wikipedia
- Uses same basic EAV model that RDF is built upon
- <u>Massively</u> interlinked



Rationale

- PLM system is "closed"
 - No web services / API
 - Weekly reports with flattened data (CSV)
 - Changes sent as incremental messages over ESB (ASCII/CSV/XML)
- Need to migrate from a legacy ASCII message to new XML message
 - Flat message
 - Implicit links
- We need more data, but didn't know exactly what
 - Wanted to be able to flexibly query the data leveraging the conceptual structure NOT the explicit structure in a particular serialization
- Point-to-point integration does not make sense
 - Specific mappings would have to be defined in the channel
 - Not robust and more effort to maintain



Why use Linked Data

- Consensus that a canonical model is a long term goal
- Challenge to define RDB and XML schema
 - Many classes
 - Reuse of properties across classes
 - Heterogeneous data
- Prototype with querying XML message using XSLT, XPath
 - Over 1 minute to build explicit tree structure from single 'root' item
 - A single message can contain over 1000 root items
 - Does not allow to traverse links in reverse direction
- Prototype with generating RDF (Turtle) and query with SPARQL
 - 200 ms to convert message AND run query (Jena in memory)
 - Using arbitrary length property paths



Modeling as RDF

- Bottom-up approach
- Model is already defined in source system
- Start with the instance data
- Data first, model later
 - RDF Schema is used to describe the data, not constrain
 - We didn't even get round to making the RDF Schema yet!



Mapping to RDF (1)

- Map class and property names converted to CamelCase in URIs
- Follow convention that classes start with initial caps, properties lowercase

Name	Slug	CURIE
Sales Item	SalesItem	plm:SalesItem
CEPT	Cept	plm:Cept
Orderable Part Number	orderablePartNumber	plm:orderablePartNumber
Status	status	plm:status

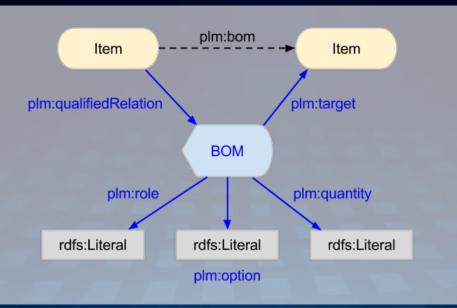
For instances we combine the class name and key id to build URI

- http://example.com/id/{class}/{id}
- Result
- http://example.com/id/salesItem/1234567890
- Guarantees uniqueness



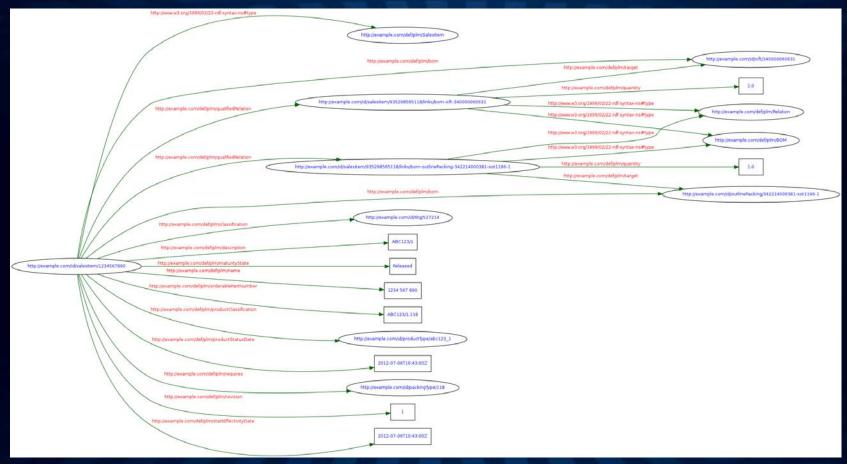
Mapping to RDF (2)

- Values do not have a datatype in the source XML
- By default map values to plain literals
- Specific match (regex) for timestamps and map to xsd:dateTime
- Could be extended in future
- Qualified links get reified





Example item description



Visualization created with W3C RDF Validator (http://www.w3.org/RDF/Validator/)



Data management

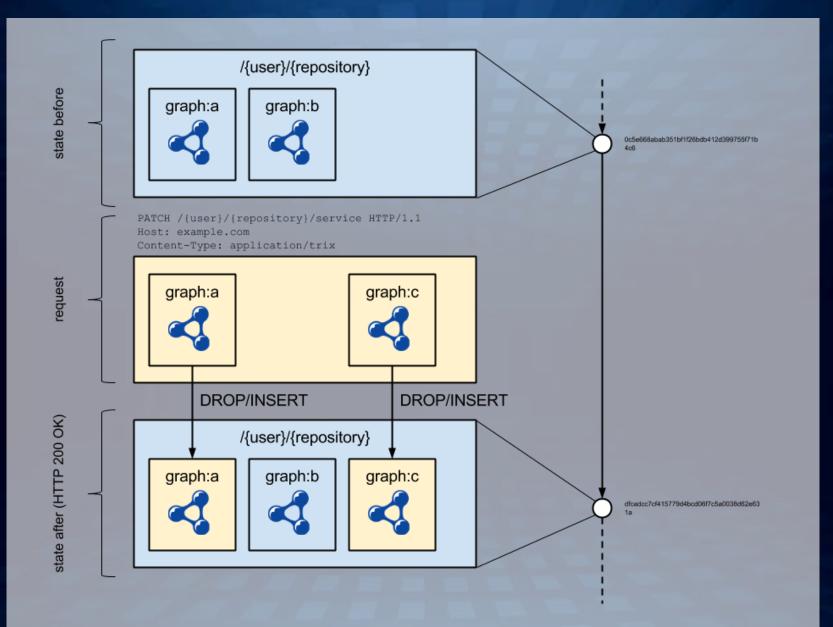
- PLM system distributes Δ updates via XML messages over ESB
- A single message contains description of multiple items
- Messages can contain overlapping content
- Simplest approach is "Graph Per Resource" data management pattern – <u>http://patterns.dataincubator.org/book/graph-per-resource.html</u>
 - Enables HTTP operations (GET, PUT) to manipulate individual
- resource descriptions



Apply changes to RDF graph store using Quads and HTTP PATCH

- We define HTTP PATCH using Quad data as equivalent to:
 - DROP SILENT operation on each named graph in payload, followed by
 - INSERT DATA operation on each named graph in payload
- Enables update of multiple resource descriptions (named graphs) in a single ACID transaction (i.e. HTTP request)
 - Same granularity as original message
 - No ambiguity about state of store, change is never partially applied
- Operation is carried out against the graph store service endpoint
 - Same endpoint as used with SPARQL 1.1 Graph Store HTTP Protocol
 - No graph or default parameter is passed with request
 - Content-Type header used to specify MIME type of the Quads format
 - application/n-quads
 - application/trix
 - application/trig
 - Quad data passed as message payload



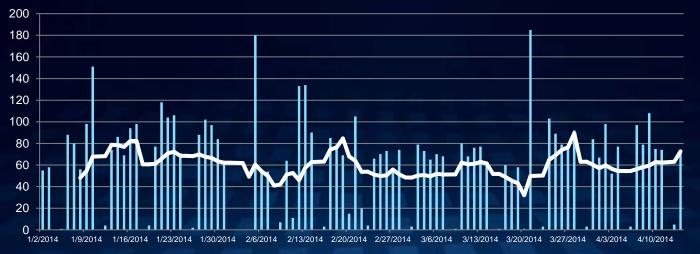




ETL pipeline

- PLM system distributes XML messages over ESB
- Generic XSLT transformation to TriX
- Load TriX to graph store using HTTP PATCH method
- Typically 3-4 seconds to transform and load a message







Facts and figures

- ~14M triples in dataset
- Describing ~0.5M items
- ~1.7M links between items

Per day	Typical	Max
Messages	60	185
Changed items	530	5,300
Sent items	14,000	77,000
Triples loaded	500,000	2,685,554



Publish as Linked Data

https:

- Used Linked Data API to make data browseable as HTML
- Also provides simple API for XML, JSON and CSV

🚽 List of sales items	×		and a second second second	B Sections Constant &	
← → Ĉ 🗋 nww.p	prod2.spider.nxp.com/de	oc/plm/salesItem			☆ 🎇 =
		laster Data _{RDF/X}	ML RDF/JSON Turtle Simple JS	ON Simple XML HTML Atom CSV	
	List of sale	es items			
	9352 985 65118 description	3 SSL21084AT/1	٩	 > List of ECOs > List of basic types > List of product types > List of main article groups > List of package outline versions 	
	type	Salesitem			
	9339 317 40113	}		On This Page 🛛 🔽	
	description type	1N4748A SalesItem	٩	> 9362 985 65118 > 9339 317 40113 > 9336 005 80113 > 9356 007 80113	
	9335 005 80113 description type	BZV85-C5V1 SalesItem	٩	<pre>> 9335 007 70133 > 9339 317 20113 > 9335 007 00113 > 9336 007 20133 > 9336 007 70113 > 9335 007 40113 > 9335 007 40113</pre>	
	8335 887 88113			Concession of the local division of the loca	
	9335 005 80113 description type	BZV85-C5V1 Salesitem		> 9339 317 20113 > 9335 007 00113 > 9335 007 2013 > 9335 007 70113 > 9335 007 70113 > 9335 007 40113 > next >	
ode.goo	gle.com	/p/puelia	-php/		



Building a Linked Data application

LINK MANAGEMENT



NXP Product Tree

- Part of main site navigation on NXP.com
 "Marketing" view on NXP product catalog
- Not a strict tree structure
- Since October 2013 is managed as SKOS concept scheme using SKOSjs

Storigs Thesearce Manage X	which its summary is in the summary of			00.0
← → C 🗈 nww.prod2.spider.nxp.com/skccjs/				\(\alpha\) =
SKOSJS				Constraint and Constr
Preset Hale			Search:	niguear (lingout
riseus res		_	and of	
NOP marteling beinomies Markeling Tree	< http://data.rep.tors/alconepta///22 G 🛢			
Froduct ree	Disader concepto :		Proformed label	
Products by Arrichon T	Pioducts to function		101525	
Ampiters II Autoristado III	L T DOUBLE AT REPORT		NOSFETS	/*
 Byolar transitions. E1 	Namour concepts		Crated	
Data convertions	Automotive MCGFETz		2013-13-01711-40-50	
Dodes D	Diander#MODFETe		Wolfed	
Identification and security in	AF power bassistors (LDHOB)		2013-19-01711-40:50	
Infertion and connective 17	RF situal-signal MCSPETs			
KOPETa	L P. Harrison Acardia		Publich Ic web	
Automotive MOSPETs ==			Yes	/ *
Clandord MOOFETa 11			Renng	
REpresentations (LDMOS)			Summer .	+
Brasdeau1/ISM =			Dominieri	
Aerospace & Dateriae				+
RF smat-signal MODPLETs 12	17			
Macha processors Microcontrollors				
Power management III				
Par E				
Substars :				
Therapora				
Products by apprication [1]				
Berkizled Emerange III Development lata III				
Mining Mappings 2				
 Notice be potentied E2 				
Obsolete branch				
Package tranch Till Midden Application Time Till				
Fine Concepts				
2	and the second s			
	a product of Salitana Research	_		

https://github.com/tkurz/skosjs

NP	Harte Abou	alain tiev	is Careers Investors	IDP Newsletter Nith Dog Conta	d Giebal / English =
 Find and Buy Products 	V Explore Applica		V Oet Support	V My100 ¹⁰	
Home Products					
All products					19 Bit produc
Arrgénes > Adde crafte > Adde crafte Adde crafte	rradio) ra banalatore store ustore banose for baol	 IJFC and in Small Land Small Land Small Land Small Land Straut Land Arsise Transmission Arsise Transmission Arsise Transmission Arsise Transmission Arsise Transmission Arsise Transmission Child Transmission	un di frincializza esteritza esteritza cita cita cita santatoritza santatoritza consectiva consectiva consectiva esteritza est	Level - Avallag version - Bufankanhreidekens - Bufankanhreidekens - Burasekter - Burasekter - Discher Burasekter - Profester - Franzens - Linder anfluende - Linder anfluende - Linder anfluende - Prake Linder - Dispersionen - Prake Linder - Dispersionen - Dispersione	Previo Intelligente + ACINEGO carbon + ACINEGO carbon - Ligneng over autorised Co- - Ligneng over autorised Co- - Ligneng over autorised Co- - Ligneng over autorised Co- - Acinet + Acinet + Acinet + Acinet - Acinet
 Imaging Procedure Voleo convertera Dodes Bandhakthing Godes FINI dodes General particle Schotter Verden pawer Schotter 	y dictare e	 Line deven Mernety data Macental Nacental NaFC & control 	i la switches lactess reader ICs lactess reader ICs lactes a social	Dartropares Transadvers Hecks processors Garcaso & audo CSP solutions Cap-Crib processors Diplain lado processors Multime ba processors Option structure	Viberes microcontrolent Eendont Angular benarers Copacitive sension P'S temperature/cofage menitors Rosational sension E-Introductive sension
na s Schothy doctes (less cap validation of the second Power Socket (lypertail Power Socket (latitudin Power Socket (latitudin Power Socket) Statituding (bode) Statituding (bode)	acturca RF) ractual() accual()		ditorers - USB 3.0 Ha chips Ha artisolers	Cars If downconverties Microcontrollers Codex/UL089- Codex/UL089- Codex/UL0 Codex/UL0 AUL07 AUL07 AUL07 AUL09	Teyrinkan • 3-Soundman Triacs (High Commutatori) • 4-Soundwart Triacs • 4-20 Thyraters • AD Thyraters • AD Thyrater Triads • DOFile
CO, CMI and signal condit General ESD protection of Application specific ESD	10/000			215-bt legary Application specific Vireleas microconnotiens	Temperature and Overload Protects Thace IV and STIE Broot ends Double demonstrations



Product placement

- Products need to be placed (linked) to product categories
- Products defined in PLM data set, categories in SKOSjs
- Different role and users to tree manager
- Decided to implement a simple application
- Use cases:
 - Find unassigned products
 - Search for products and categories
 - Add links
 - Remove links



Let's Play

- Decided to build application using Java version of Play Framework
- Approx 4-6 weeks to develop and test
- Application makes queries against graph store direct from client browser
 - Using SPARQL 1.1 Federated Query
 - Originally by populating pre-defined SPARQL templates and executing these against SPARQL endpoint
 - Now queries are stored in database and exposed as an API
 - Initial variable bindings can be passed as request parameters
- Results are in SPARQL Query Results JSON Format



SPARQL Query Results JSON Format

- Simple tabular results format for SELECT and ASK queries
- > Content-Type: application/sparql-results+json
- Example response

```
"head": {
    "vars": [ "btn", "desc", "csi", "psi", "status", "statusDate" ]
},
"results": {
    "bindings": [
            "btn": { "type": "literal", "value": "BZL3615AHN" },
            "csi": { "type": "literal", "value": "No" },
            "psi": { "type": "literal", "value": "No" },
            "status": { "type": "literal", "value": "Development" },
            "statusDate": {
                "type": "literal",
                "value": "2014-02-03",
                "datatype": "http://www.w3.org/2001/XMLSchema#date"
        ... more bindings ...
```



Example 1: Get unassigned products

SELECT query with no initial bindings

GET /nxp/marketing-tree/pp_get_unassigned_products.srj HTTP/1.1

btn	desc	csi	psi	status	statusDate
BTH3415TIO		No	No	Development	2014-02-03
BTM4500TIO		No	No	Development	2014-01-31
CHB3131		Yes	No	Development	2014-03-04
CHV7072		No	No	Development	2014-03-12
CHV7075		No	No	Development	2014-03-12
CHV8I1		No	No	Qualification	2014-01-23
CHV8M1		No	No	Qualification	2014-01-23
CHV8O1		No	No	Qualification	2014-01-23
CMD8H21MT-160B		No	No	Development	2014-04-07
CMG6H22MT-180Q		No	No	Production	2014-03-20



Example 2: Get products filtered

- SELECT query with a single binding for search string
- Initial bindings passed as parameter \$searchString
- Literal values enclosed in quotes "2n7002"
- Parameter name and value URL encoded

GET /nxp/marketing-

tree/pp_get_products_filtered.srj?&%24searchString=%222n7002ck%22
HTTP/1.1

btn	desc	csi	psi	status	statusDate
2N7002CK	60 V, 0.3 A N-channel Trench MOSFET	No	No	Production	2011-10-28



Example 3: Link product to category

- INSERT query with binding for product and category
- Initial bindings passed as parameters \$productId \$categoryId
- Literal values enclosed in quotes "2n7002"
- Parameter names and values URL encoded

GET /nxp/marketing-tree/pp_link_product_to_category.srj? %24productId=%22BFU915F%22&%24categoryId=%22208%22 HTTP/1.1



				a second s	
💗 Home	×				
← → C 🗋 nww.pr	od2.spider.nxp.con	n/produ	uctplace	ment	
	INT ⊦ Home				
Enter (part of) category id/	name or product name	e: unass	signed	S	earch
	Unassi	gned prod	ducts		
Product name 💠	Descriptive title 💠	CSI \$	PSI 💠	Status 🔶	Status date 🔅
ASL3415SHN		No	No	Development	2014-02-03
ASL4500SHN		No	No	Development	2014-01-31
BGA3131		Yes	No	Development	2014-03-04
BGU7072		No	No	Development	2014-03-12
BGU7075		No	No	Development	2014-03-12
BGU8H1		No	No	Qualification	2014-01-23
BGU8L1		No	No	Qualification	2014-01-23
BGU8M1		No	No	Qualification	2014-01-23
BLC8G21LS-160AV		No	No	Development	2014-04-07
BLF6G22LS-180PN		No	No	Production	2014-03-20
First Previous Next	Last Page: 1	G	0 F	Rows: 10 🔻	



✓ Product 2N7002CK × ✓ → C nww.prod2.spider.nxp.com/productplacement/product/2N7002CK



PRODUCTPLACEMENT Product 2N7002CK

Name:	2N7002CK
Descriptive title:	60 V, 0.3 A N-channel Trench MOSFET
Published to web:	Yes
CSI:	No
PSI:	No
Status:	Production
Status date:	2011-10-28
SSPD:	small signal MOSFET
Category: (part of category id/name	e) Filter

Categories with product 2N7002CK

Category name \$	Category ID 🗘	TriMM ID 🗘
Automotive MOSFETs	130	50933 🗙
Standard MOSFETs	67	48014 🔀
First Previous Next Last Page:	1 Go Rows: 10) ▼

Categories without product 2N7002CK

Category name \$	Category ID 💠	TriMM ID 💠
(De)multiplexers, data and clock recovery, limiting amplifiers	208	42486 🔶
0-1600 MHz (HF / VHF / ISM)	672	53511 🔶
0-500 MHz (HF / VHF / ISM)	1254	71254 🔶
0.4 - 1.0 GHz transistors	1190	42804 🕂
1 GHz - 2 GHz (L band)	1805	71805 🛖
1 GHz -> + 2 GHz (L band)	1699	71699 🛖
1 GHz -> + 2 GHz (L band)	654	30921 🔶
1.3 - 1.7 GHz transistors	1417	71417 🔶
1.8 - 2.0 GHz transistors	1191	42805 📥
10-500 MHz (HF / VHF / ISM)	673	53512 🕂
First Previous Next Last Page: 1 Go Ro	ows: 10 🔻	

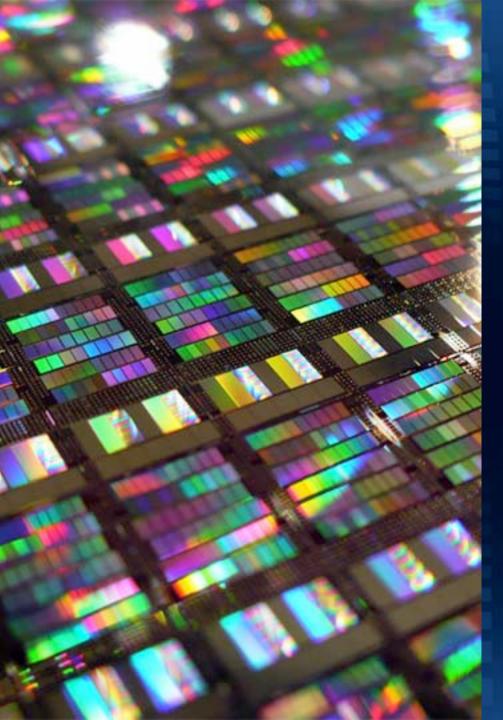


Lessons learned

- Building a simple Linked Data application is easy
 SPARQL Federation is very useful
- Stored queries is a good approach to expose API

 Front end developer doesn't see SPARQL
 Works for read and write





Publishing Linked Data

LINKED DATA API



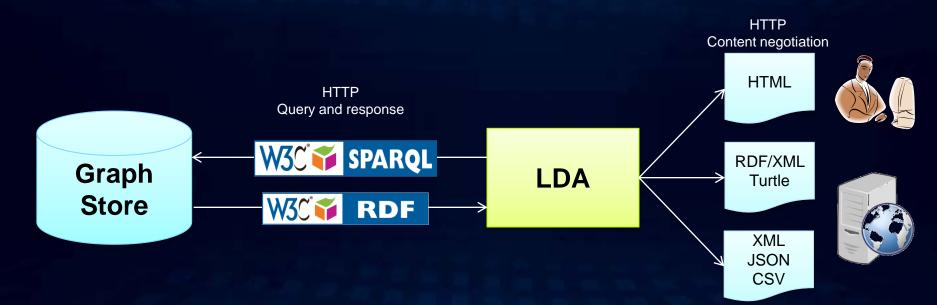
Linked Data API (LDA)

- LDA defines a vocabulary and processing model for a configurable API layer intended to support the creation of simple RESTful APIs over RDF triple stores.
- The API layer is intended to be deployed as a proxy in front of a SPARQL endpoint to support:
 - Generation of documents (information resources) for the publishing of Linked Data
 - Provision of sophisticated querying and data extraction features, without the need for end-users to write SPARQL queries
 - Delivery of multiple output formats from these APIs, including a simple serialization of RDF in JSON syntax



LDA Architecture

- Use W3C web standards RDF, SPARQL for portable solution
- Work with any RDF graph store with only minor configuration





LDA is open source

- Open source specification: <u>http://code.google.com/p/linked-data-api/</u>
- Open source implementations:
 Puelia (PHP): <u>http://code.google.com/p/puelia-php/</u>
 Elda (Java): <u>https://github.com/epimorphics/elda</u>

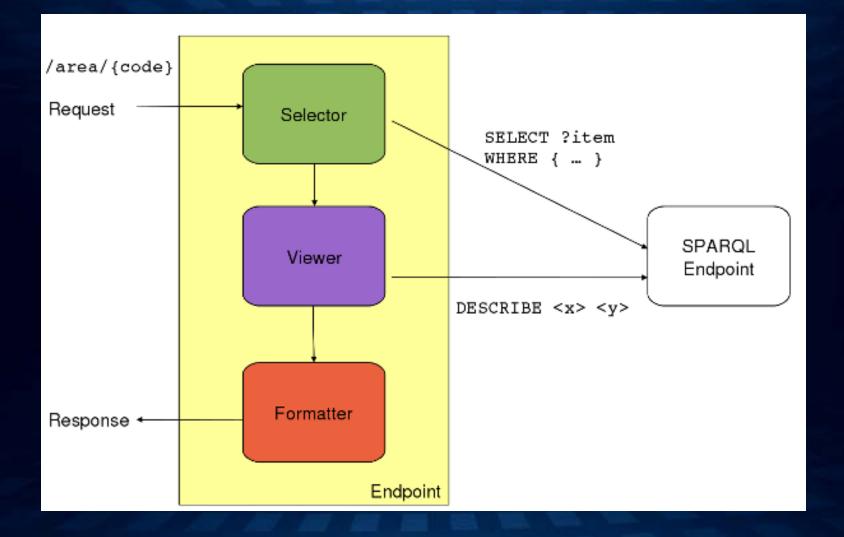


LDA Processing Model

- Identifying an Endpoint -- GET request made to a particular URI is mapped to an Endpoint that describes further processing logic
- Binding Variables -- the API creates a number of variable bindings based on the structure and parameters of the incoming request, and the Endpoint configuration. These variables as well as the available API configuration and request metadata describe the Context for the execution
- Selecting Resources -- the API identifies a sequence of items who properties are to be returned. Usually this will be based on a Selector that describes how to identify a single item or an ordered list of resources, in concert with the available bindings
- Viewing Resources -- the API retrieves the desired properties of the identified resource, constructing an RDF graph of the results. This process is described by a Viewer that identifies the relevant properties of the resources
- Formatting Graphs -- the API identifies how to serialize the resulting RDF graph to the client. This process is defined by a Formatter



LDA Processing Model





LDA configuration

- LDA config files are declarative description of the API
- Describes
 - List and item endpoints
 - Selectors
 - Viewers
 - Formatters
- Written in Turtle
- Read by application at run time



