2008

RAMLET implementation study report

Willem G. Kraan

University of Bolton, w.g.kraan@bolton.ac.uk

Digital Commons Citation

http://digitalcommons.bolton.ac.uk/iec_reports/28
RAMLET implementation study report
Wilbert Kraan, JISC-CETIS
17/09/2008

Table of Contents

1 Introduction .................................................................................................................. 1
   1.1 The interoperability problem ................................................................................. 2
   1.2 The IEEE RAMLET project .................................................................................. 3
2 Data integration patterns and RAMLET ...................................................................... 4
   2.1 RAMLET ontology characterisation .................................................................. 4
      2.1.1 Data integration approaches: local as view versus global as view ............... 4
      2.1.2 Ontology architecture: RAMLET as a hybrid multi-ontology ................. 5
      2.1.3 What RAMLET describes ............................................................................ 6
   2.2 Ontology application architecture patterns ...................................................... 7
      2.2.1 The central triple store pattern .................................................................. 8
      2.2.2 The distributed query pattern ................................................................... 13
3 Ramlet Use cases and JISC practice ........................................................................... 15
   3.1 Scenario one - The Scholarly Cycle ................................................................. 16
      3.1.1 The Scholarly Cycle scenario and the central triple store pattern ............. 18
      3.1.2 The Scholarly Cycle scenario and the federated query pattern ............... 19
   3.2 Scenario two - The all-round academic ............................................................. 21
      3.2.1 The all-round academic scenario and the central triple store pattern ....... 22
      3.2.2 The all-round academic scenario and the federated query pattern .......... 23
   3.3 Scenario three - Complex Objects in Teaching and Admin ............................. 24
      3.3.1 Complex Objects in Teaching and Admin and the central triple store pattern ......................................................................................... 24
      3.3.2 Complex Objects in Teaching and Admin and the federated query pattern ................................................................................................. 26
   3.4 Federated query versus the central triple store pattern ..................................... 27
4 Implementation technology options ........................................................................... 28
   4.1 Central triple store pattern options ..................................................................... 28
      4.1.1 Transform system ...................................................................................... 28
      4.1.2 Triple store ................................................................................................... 28
   4.2 Federated query pattern options ......................................................................... 29
      4.2.1 Agent ............................................................................................................ 29
   4.3 The compositor and other shared infrastructure ............................................... 30
5 Recommendations .................................................................................................... 30

Research made possible by a JISC grant

© University of Bolton

Creative Commons Attribution-ShareAlike 3.0 Unported License
1 Introduction

In this report, an outline will be given of the implementation options for the RAMLET ontology, which is a model of how to relate various standard complex object formats to each other at the level of meaning. The report will take the form of a set of recommendations, that have been derived from a set of application architecture patterns, as related to a set of relevant scenarios and use cases, and applicable semantic integration technologies.

At each of these stages, the relevant architectural models will be presented as Archimate diagrams (Lankhorst, 2004). The main reason for doing that is to be able to relate the architectures to each other at various levels; from a broad business scenario to a detailed application component. It is also hoped that the diagrams can be contrasted and compared to other Archimate diagrams, either in the emerging JISC Innovation Knowledgebase, or elsewhere. The full diagrams are also included as appendices.

First, though, the RAMLET ontology, as well as semantic data integration in general, will be situated in the wider field of interoperability. A short outline of the IEEE RAMLET project will follow.

1.1 The interoperability problem

With rapidly increasing amounts of data of increasingly different kinds being captured by an increasing array of systems, the question of how to exchange all that information remains open. The complex content objects that RAMLET addresses, for example, barely existed before the multimedia CDs of the mid 1990s, or the first structured zip archives of the late 1990s. Now there are large amounts of them in various silos in a plethora of formats. Few of these are likely to achieve significant use without a means of making them accessible across formats. I.e. without interoperability, they could remain un-used.

In simple terms, enabling interoperability means that both transport and data need to be aligned. Since the advent of the internet in general, and the web (http) in particular, this is much less of an issue than it would otherwise be. Especially now that there is a fairly strong emergent consensus around the use of REST APIs (Lacey, 2007), the mechanics of swapping data is becoming well-understood and tractable.

The data itself, though, is another matter. In order to exchange information between different systems, interoperability needs to be established at roughly three different levels:

1. syntax; the format of the data
2. structure; the way the data is arranged, usually in some sort of schema

3. semantics; the meaning of the data

It could be argued that there is a fourth level: pragmatics, or the negotiation of meaning of the data / information in its context of use. This does not appear to be a widely used concept in ICT, however, and it could be argued that – in ICT systems – the meaning of data is so directly context-bound that a distinction between semantics and pragmatics is not necessarily helpful.

Of the various levels of interoperability, the widespread adoption of XML has meant that incompatibilities in the syntactic form of data has become much less of an issue. While recent web applications sometimes favor other representations such as JASON for ease of programming, translating between them is fairly easy programmatically (Goessner, 2006).

As noted by one of the 'fathers' of XML, though, structural incompatibilities in how similar information gets recorded in XML structures, and differences in what those structures really mean, suggest that simply inventing more XML based languages does not solve the interoperability problem (Bray, 2006). While Bray's exhortation to just re-use a few existing languages is laudable, the fact is that there are already quite a few such languages in active use, and communities are likely to continue inventing more.

What's needed, then, is a means of abstracting over not just syntax, but also structure and semantics. Which is where the W3C's semantic web stack comes in.

While incompatibilities in XML structure can be resolved using XSL transforms, the development effort is often non-trivial, particularly when more than two XML languages need to be translated. Without a common intermediate format, any change in one language, quickly leads to a combinatorial explosion of changes in the transforms to all the other languages. Also, subtle interactions between structure and semantics can easily creep in at either at the time the XML languages were invented, or in use.

RDF, by contrast, has a single, fixed structure, whatever its application. People can and do invent their own RDF vocabularies, but at least they all have the same (triple) structure. As a consequence, various RDF applications can share a lot, if not all of the software infrastructure they require. Also, since RDF is canonically represented in XML, it is fairly easy to use it as the common intermediate format in a set of transforms between various XML languages.

Which leaves the semantics of all that data. While it is possible to ascribe any kind of
meaning to any resource in RDF, this is not much use until it can be related to other vocabularies. For that purpose, both RDFS and OWL provide agreed vocabularies to define such relations. Note that such bridging doesn’t entirely solve the semantic interoperability issue. No transform can insert meaning that isn’t predictable or defined in the source. That is, if systems that use vocabulary B expect non-predictable element Y, and source vocabulary A doesn’t have element Y, then no transform between A and B can introduce Y.

Still, while the original vision that led to the development of this semantic web stack (Berners-Lee, Hendler, & Lassila, 2001) remains firmly speculative and/or futuristic, it’s designed ability to abstract over transport, syntax and structure differences, while preserving as much of a format’s semantics, suggests that it should advance the current state of the art in interoperability. Whether that promise is kept in practice depends largely on the maturity of the tools that support XML, RDF and RDFS / OWL over REST as well as the architecture that aligns them. Hence this report.

1.2 The IEEE RAMLET project

The IEEE Learning Technology Standards Committee has been supporting the work on RAMLET, the Resource Aggregation Model for Learning Education and Training. The model addresses the need for common complex object formats for content from the multimedia, library, web and e-learning worlds to interoperate. To do that, RAMLET is a conceptual model, expressed as a human readable table and a set of machine readable OWL ontologies that specify how these aggregation formats map via one core model. The RAMLET model is feature complete, but has not been implemented yet.

The reason why it is the Learning Technology Standards Committee of the IEEE that is sponsoring the RAMLET work lies in a recognition that content for learning and teaching comes from a variety of domains. For that reason, there is a clear need to establish interoperability between the content aggregation or complex object formats that prevail in adjacent domains.

In order to allow the interoperability established by RAMLET to scale, the model does not just map pairwise between formats – as one would in a cross-walk –, but maps the various aggregation formats to one core ontology. Moreover, in order to preserve as much of the intended meaning of a content aggregation as possible when transforming from one format to another, the core RAMLET ontology accommodates all components of all aggregation formats that it maps. Put differently, the RAMLET core ontology is a superset of all the
formats it maps. This way, any mapping from one of the constituent aggregation formats to a RAMLET core representation is lossless.

Though there is no hard limit to the amount of aggregation formats that could be mapped to future versions of RAMLET, the current version has mappings from the following aggregation formats to the core ontology:

- **IMS Content Packaging** from the teaching and learning domain
- **Metadata Encoding and Transmission Standard (METS)** from the library domain
- **MPEG-21 Part 2: Digital Item Declaration Language (DIDL)** from the multimedia domain
- **IETF Atom syndication format** from the web

The RAMLET working group is currently working on extending the IETF Atom mapping to include the OAI-ORE\(^1\) extensions of Atom. That way, serialisations of OAI-ORE can be mapped to all the other aggregations included in RAMLET.

The mappings from these formats to the core ontology are also expressed in OWL. Beside the functional relationships between the components of the aggregation formats to the core, these mappings also capture the structural or syntactic constraints of the formats themselves. As a result, the RAMLET set of ontologies is an abstract expression of all the information required to reliably transform a (part of) an aggregation instance of one constituent format to another.

Because of its abstract nature, the RAMLET model can be used in a number of different technologies, either directly, via automatic conversion or via a developer's interpretation.

## 2 Data integration patterns and RAMLET

Though RAMLET is open to a number of different implementation technologies, it does have a number of salient characteristics in its general approach to data integration. These, in turn, relate to a number of general implementation patterns that will guide the remainder of the report.

### 2.1 RAMLET ontology characterisation

#### 2.1.1 Data integration approaches: local as view versus global as view

Unusually for ontology-based data integration, RAMLET assumes a global-as-view

---

\(^1\)Open Archives Initiative Object Reuse and Exchange
approach. That means that it is derived from the source schemas (i.e. the constituent aggregation formats outlined in section 1.2), and is a proper union of them. This way, RAMLET can function as a single query model over any number of source repositories that contain instances of the constituent aggregation formats. Put differently, in order to get at the information contained in a large number of different complex content objects, you’d only have to know the RAMLET model in order to ask the question, and do something with the returned information.

The disadvantage of such an approach relative to the local-as-view (LAV) alternative is that it puts all the hard work on the mappings from those constituent aggregation formats to the core model (Lenzerini, 2002). Also, adding any new source schemas to the global model means having to change that model. Local-as-view approaches don’t have those drawbacks as the global is essentially the sum of a set of (partial) views over the source schemas. Since the global schema or model is defined independently of the source schemas, adding more source schemas does not change the global model. Resolving any query over the global schema is more difficult to formulate and execute, however.

The IEEE working group did consider what would have been a LAV approach, but rejected it out of several considerations. Most importantly, such a model would have effectively become another complex content object standard, and the general effort was to reduce fragmentation in that area, not increase it further. Furthermore, as Lenzerini’s (2002) review also points out, a LAV approach is most effective when the global schema is comparatively stable and particular (e.g. a business ontology), but the sources changeable and diverse. A GAV is more appropriate for stable sources and a global model that needs to be generic.

Since RAMLET is meant to address interoperability between different communities, rather than make the constituent formats available to just one community and its uses, a GAV approach was thought more appropriate. Also, while there may be future complex content object formats that would require changes to the RAMLET model, the current set of formats is relatively stable and not likely to change drastically because of the considerable installed base of each of them. Furthermore, additional formats may well add a few more RAMLET components, but it is unlikely to change the existing set, or its internal relations.

Lastly, it should be noted that while the LAV versus GAV distinction is useful for the purposes of data integration modelling, the distinction seems rather less sharp when applied to systems based on triples rather than relational tables. That is, the difference
between local to global schemas that motivates the distinction is largely one of structure. That is, LAV and GAV are mostly a means of relating keys, values and tables (in database terms) or elements, attributes and their values (in XML terms) from one schema to another. In an RDF world, where the assumed structure is always a triple, such fixed relations between elements is either of no consequence or just another piece of information that is as mutable as any other.

As a consequence, RAMLET's model is based on the function of the elements in a given aggregation while their structural relations (in so far as any can be generalised from the constituent formats) are described in separate property constraints. The structural information is there if it needs to be exploited, but the model is built more directly for element to element mapping.

2.1.2 Ontology architecture: RAMLET as a hybrid multi-ontology

As noted in the previous section, the GAV approach does depend rather strongly on the mapping from the source to the global schema or model. This has consequences for the choice of ontology architecture and its implementation, because a lot of the effort of making a working GAV based system lies in the business logic of the source to global mappings.

At this level, the choices are between how the source and global models are distributed over specific ontologies- both in the sense of abstract namespace and discreet ontology file. For the purposes of ontology based information integration, three broad patterns have been identified: single ontologies, multiple ontologies and a hybrid approach (Wache et al., 2001). The advantage of combining source and global models in a single ontology is its simplicity. Disadvantages include the difficulty of combining more heterogeneous sources, and the fact that any small change in any of the sources affects all the rest.

Multiple ontologies that model each source make that ontology management job easier and can also easily accommodate new sources. Modelling the global model is fairly difficult, however, and so is maintaining consistent relations between the sources and the global model in each ontology. Comparing sources, as a consequence, isn't easy either.

For that reason, people increasingly use hybrid approaches, where the sources are each modelled in their own ontologies, but where those ontologies are linked together using a common vocabulary or global ontology (Wache et al., 2001). In what one could call a 'light' hybrid model, what is shared between the source ontologies is a common descriptive
vocabulary to help comparability. In a heavier hybrid model, the global model is an ontology in its own right, which is explicitly related to the sources.

RAMLET is example of the latter, in that its core is a general ontology that covers all the elements of each constituent content format as a refinement. That is, each constituent format is modelled in its own ontology (i.e. the 'mappings'), and all these format ontologies are then imported into the core ontology, where they are modelled as subclasses (i.e. refinements) or core ontology classes.

This arrangement does have the disadvantage that it makes the core ontology relatively large, and it also means that any genuinely new type of element from the content format requires a new core ontology class. The separation, though, allows a degree of modularity in the creation and management of the various models, and it also makes the models reusable. Most importantly, the separation best accommodates the refinement / mapping mentioned earlier.

### 2.1.3 What RAMLET describes

That refinement / mapping approach is what Allemang & Hendler (2008) call the amalgamation pattern of data integration with RDF(S). The challenge when integrating data such as the complex content object formats that RAMLET tackles, is to find as much commonalities between the various elements as possible. That commonality, however, rarely extends to full equivalence.

In Figure 1, for example, the IMS Content Packaging *Manifest* element is quite comparable to the IETF Atom *Feed* element: both serve to define the root of a content object, and serve to carry its major divisions and identifying elements. Declaring them equivalent using something like OWL's sameAs assertion or making them RDFS subclasses of each other might seem a shortcut to modelling their 'sameness' for the purposes of integration. The
trouble is that they are dissimilar in other contexts. Structurally, for example, *Manifest* is recursive, while *Feed* is not.

To assert the right degree of similarity, it needs to be put in the right context. In the amalgamation pattern, this is achieved by declaring a relatively 'light' superclass for all those elements that are comparable for the purpose of the integration ontology. As a consequence, it is fairly easy to infer that instances of *Manifest* and *Feed* make good candidates for a *METS* element, without leading to the conclusion that all their associated structural constraints would map straight across as well.

All of this leads to what it is about complex content aggregations that RAMLET describes. As noted, that is the function that the structural elements of these formats have in a complex content object. To stay with the example from Figure 1, RAMLET's *TopNode* intension is currently defined as “The highest class level of an aggregation defined by an aggregation format.” (IEEE LTSC, 2008)

Describing other aspects is just as possible, and where explored earlier in the lifetime of the IEEE LTSC project. For one, there is the subtly different aspect of the structural relations of the elements, as already discussed in section 1.1. The other possibility is to try to model various underlying models such as the (pedagogic) intent of the author, informational structure or the general processing of complex objects assuming particular computational approaches such as procedural or object oriented processing. The advantage of modelling such aspects is that they may well lead to much more compact and persistent models. Not all quirks of an XML binding need to be modelled to capture concepts such as “test”, or an informationally atomic, standalone section with its own navigation, for example. Nor would parts of a complex object that are defined by informational purpose be affected by changes in the packaging format.

For that aspect of the informational purpose of complex object content, an ontology has already been demonstrated (Verbert, Duval, Meire, Jovanovic, & Gasevic, 2006). This ALOCoM ontology (Verbert & Duval, 2004), however, is intended to facilitate the re-use of learning content at various levels of granularity. Trying to use such an ontology for the purpose of complex content object data integration, would lead to a couple of issues. For one, not all the constituent formats of RAMLET are intended for teaching and learning purposes, which means that they could have very different informational purposes that may not map easily. For another, establishing informational structures of such a type requires human judgment. Not a problem when there is an infrastructure in place that
captures such information at authoring time, but more of an issue when the goal is to establish a scalable data integration solution for a large volume of heterogeneous legacy content.

Clearly, RAMLET is not a comprehensive model of complex content objects. For purposes other than data integration, other ontologies are possible, and could be as complementary to RAMLET as ALOCoM. But to fulfil its purpose, RAMLET has had to focus on modelling just those aspects that get typically described in the standards documents that accompany the XSD Schemas of complex content objects, and relate them to each other.

2.2 Ontology application architecture patterns

The last decade or so has seen a steady trickle of new semantic technology releases such as the completion of, first, RDF (Lassila & Swick, 1999), the OWL ontology language (Bechhofer et al., 2004) and the SPARQL query language (Prud’hommeaux & Seaborne, 2008). As a consequence, the tools that implement these technologies have steadily matured, which is beginning to indicate an interesting progression in the architecture in which these technologies were conceived to function.

In the original vision for the semantic web (Berners-Lee et al., 2001), the main idea was that intelligent agents would roam over RDF data sets like browsers over a web. Hence the metaphor.

That hasn't materialised (yet) for a number of reasons that are beyond the scope of this report. In the mean time, the main technology to manage RDF datasets – triple stores – were developed and led to a more scaled down vision and architecture. Rather than assume that all data will be available as triples over the web, the central triple store architectural pattern 'sucks' data of different types from various places into a database optimised for RDF processing. The replicated data is further processed with specialised tools in a 'pure' semantic technology environment.

The advent of SPARQL and other query languages, however, has made it possible to design a more distributed architecture. Rather than replicate the data, and contain in it a single place, this architectural pattern leaves data where it is stored, but queries the databases using semantic technologies. There maybe further processing post query, but no central data storage.

It is the central triple store and distributed patterns that we'll be exploring next.
2.2.1 The central triple store pattern

The central triple store pattern is pretty much the default ontology application architecture (Allemang & Hendler, 2008), and is outlined in Figure 2. It is premised on the idea that most interesting data is very heterogeneous in nature, and spread all over. The first step, then, is to convert the structure of these databases, spreadsheets, webpages and XML documents to RDF (either in RDF/XML, N3, turtle or other formats). This will enable the data to be uploaded into a triple store. Some basic manipulation and querying is possible at this point, but hindered by the fact that any such operation requires in-depth knowledge of all the source data. That is, in order to construct, say, a combined calendar out of a RDF-ied vCal file and a spreadsheet with some dates, you'd have to know the names and meaning of each of the relevant elements of those two sources before you could formulate a query that returns results from both.

Which is where an ontology comes in, because that would specify how these source would relate to each other in terms of a more general domain model. Before such a relation can become operational over the full data set, however, the individual instances need to be associated with the classes identified in the ontology. Put differently, if an instance of source class A and an instance of source class B are both subclasses of a domain ontology class C, then the class relation would be explicit in the ontology, but the relation of the instances to the domain ontology needs to be inferred. Once that’s done, queries can be constructed purely in terms of the domain ontology, isolating who or what is doing the querying from the details of the source data.
In order to determine what an application of the central triple store pattern to a RAMLET implementation architecture could look like in detail, we'll first examine the source data transformation components of such an architecture. This will be followed by the triple store components, and concludes with the complex object composition components. At each stage, constraints that are determined by the nature of the RAMLET ontologies will be taken into consideration, as are some relevant lessons learned from other implementors of similar architectures.

In that regard, one notable aspect in Scott Marshall & Prud'hommeaux' (2008) as well as Cowan's (2008) reports is the use of custom programs rather than XSLT to convert source data into triples. In the case of the former, that is partially explained by the fact that the datasets are large and are typically exported from relational databases in tab delimited text. In Cowan (2008), some of the source data are feeds, which are easier to parse and write out as triples with the Rome feed parsing library (ROME Project, 2008) than with XSLT.

Nonetheless, as Figure 3 shows, RAMLET may still be better off with XSLT as the main
means of transforming source data into triples. The reasons are that the complex content objects’ use XML as the default exchange format for their aggregation definition documents. Furthermore, the RAMLET ontology mappings explicitly models all the elements and attributes of each complex object format, which should make the transform fairly straightforward. Finally, there are unlikely to be any readily use-able, general purpose parsing libraries available for any of the formats other than IETF Atom. Custom coding them would introduce a potential maintenance headache, compared to maintaining a coherent set of XSLTs that provide consistent output.

Other aspects to note in Figure 3 are the wide variety of input interfaces, which reflects the requirements of the range of communities that could make use of RAMLET (see section 3).

Another notable aspect is the asset store; a peculiarity of processing complex objects. Though data integration focusses entirely on the manipulation of the Aggregation Description Documents (A.D.Ds), there are still the digital resources themselves to manage as well. Since triple stores are unlikely to be optimal storage mechanisms for the very diverse and often rather large assets, a separate asset store would be more optimal. To maintain the connection between the structure defined in the A.D.D.s and the assets, URLs could be introduced by the transform function.

![Figure 3: RAMLET central triple store architecture, data transform components detail](image)

With regard to inferencing, Scott Marshall & Prud’hommeaux’ (2008) report on ontology
merging in the biomedical sciences points out the utility of keeping inferred triples in a separate graph. That way, load and query times decreased, and changes in the ontology became easier to manage. As we'll see in section 4, reasoners tend to be quite specialised and triple stores and toolkits can easily access them as webservices. For those reasons, both the inferencing service and the resulting triples are modelled as separate components in Figure 4. As a consequence, it may well be the case that the SPARQL queries that will construct the output will run primarily over the inferred graph rather than the format-specific mapping triples. That is, queries that call for instances of the core ontology will only find matches in the inferred instances, not the source mapping instances.

Also modelled separately is the SPARQL service provider. This is partially because the practice in tools seems to vary, with some triple stores providing such querying capability natively, and other relying on general semantic web toolkits. As with inference engines, the capabilities of different SPARQL service providers differs, and being able to mix and match could be an advantage.

In terms of output, RAMLET is notable for the fact that output of the data integration process is not directly meant for human consumption. In cases such as those outlined by Gilman et al (2008), the output could still be XML, but even then it is used directly for
rendering by a single known system.

For that reason, the composition stage of the RAMLET data integration process is comparatively complex. Assets need to be fetched from the asset store, and combined with aggregation description documents into packages. As can be seen in Figure 5, IETF Atom is the exception to this rule. For that format, the aggregation definition document in XML that is the output of the SPARQL query process will be sufficient in most cases. As with the transform stage of the process, a variety of content object export interfaces are likely to be required by the various use cases.

\[\text{In Atom it is possible to exchange assets, not just links to them. Unlike the other formats, though, Atom encodes assets and includes them inline in the aggregation description documents. In those cases, it is conceivable that the assets will get stored into and retrieved from the triple store. Whether performance and storage considerations make that viable is an open question.}\]
2.2.2 The distributed query pattern

While the central triple store is robust and proven, it also has a number of disadvantages. The most important one of those is the same as the disadvantage of conventional
datawarehouses: data needs to be duplicated, and therefore quickly gets stale or out of sync. A parallel data store also means a virtual duplication of infrastructure, which means greater complexity in system management.

Leaving the data where it is, and query it in a native language when necessary, sounds attractive for those reasons. The distributed query pattern sketched in Figure 6 is therefore gaining popularity. In addition to the reasons cited above, King (2008) also reports that splitting a dataset over four triple stores rather than keeping it in one can lead to a doubling in load and query performance. He also contends that scaling the total data set is easier, data lifecycle management is more tractable, and it is easier to determine the provenance of various data. These research observations also appear to hold in an enterprise-wide production environment (Gilman et al., 2008).

When applied to the RAMLET case, the main differences with the central triple store pattern are in the transform and triple store phases. The content object composition phase is largely the same as outlined in Figure 5.

Though the architecture detail of Figure 7 shows the transform and query agents separately for each source format store, this doesn't need to be implemented that way. As we'll see in 4, it is possible for each of these agents to be implemented with the same applications on a single server. They are different applications logically, however, hence the separation.

Compared to other distributed query architectures, another notable feature is that the first step in the process after a native query in the source repositories is to convert the aggregation description documents using XSLT. Unlike cases such as the enterprise integration architecture outlined by Gilman et al (2008) or the distributed triple stores described by King (2008), the main data in the RAMLET case is ultimately exchanged in XML, not database tables or triples.
3 Ramlet Use cases and JISC practice

Though it could be argued that an implementation architecture should start with requirements (The Open Group, 2007), there are a number of other considerations that modify the prioritisation of technologies and requirements in practice. For one, ontology-driven semantic integration is a relatively new technology. Functionality is therefore not just a given, but a potential driver for new practices. For another, while technology stacks for architectural patterns other than the ones outlined in chapter 2 could meet some requirements better in theory, they’re not currently available in practice. Nonetheless, whether the centralised triple store or the distributed query pattern outlined in the previous section would be the most applicable solution to an RAMLET implementation architecture for the JISC community depends on how well each architecture addresses known JISC community use scenarios.

As a recent survey of digital repositories and archives in the JISC community indicates (Abbott & Anderson, 2008), the sheer number and diversity of digital repositories is too large to describe comprehensively. Gathering a representative sample of use scenarios is therefore somewhat beyond the scope of this study as well. Fortunately, recent work on repository APIs has articulated three wide ranging, representative scenarios (JISC Common Repository Interfaces Working Group, 2007) to guide development. Aligning the RAMLET implementation architecture with these scenarios, therefore, both re-uses relevant survey work and also eases integration with directly adjacent development work.
Since they are so wide-ranging and general purpose, the Common Repository Interfaces Group (CRIG) scenarios are defined at a relatively abstract level. In order to relate them to application architectures, more specific use cases are needed that detail how the general processes and actors of the scenarios interact with specific services and components. Fortunately, the RAMLET ontology development itself was guided by a set of seven use cases (IEEE LTSC CMI Working Group, 2005). These use cases span the specificity of the application architecture patterns and the generality of the CRIG scenarios in that they do name specific components and operations, but do not detail the internal schematic of data transformation processes themselves.

The application architecture patterns from section 2.2, the RAMLET use cases and the CRIG scenarios do not merely complement each other in their levels of modelling abstraction, they also provide a useful triangular validation to each other. The scenarios, for example, provide models of use that ground the use cases and, by extension, the architecture patterns. But the patterns and the use cases also indicate the viability of the scenarios in actual application architectures.

Since there are two architecture patterns, three scenarios and seven use cases, there are a theoretical forty-two different combinations. In practice, quite a few combinations are either not possible, or not very enlightening. For example, combining a scenario that doesn’t involve direct authoring of complex objects, with a use case that depends on such an operation over a RAMLET derived format with a pattern that doesn’t have such a format in store just doesn’t work. For that reason, the following analyses combine the three scenarios with the two architecture patterns each, and the most relevant use case for the scenario – pattern combination.

### 3.1 Scenario one - The Scholarly Cycle

The Scholarly Cycle scenario, as the name implies, is a mostly process-driven scenario, with four relatively independent phases. The scenario starts with a reading phase by a researcher with the role of principal investigator, using some search service. It consists of a resource discovery sub-process, and then some reading. This is followed by an analysis phase, which encompasses the comparing of data, analysis of the principal investigator’s own data using a specialised service, and the discussion of the results with other researchers. The writing phase continues with a draft of an article by the principal investigator, and more use of the collaboration service. It ends with the submission and review of the article to a journal via some document management service. Finally the
publish phase involves the tracking of citation and other use data from a variety of sources, and the publication of versions of the work for publicity, the Research Assessment Exercise (RAE) and for learning & teaching.

3.1.1 The Scholarly Cycle scenario and the central triple store pattern

The most immediate phase where RAMLET is relevant in the scenario is the final, writing and publication phases. It is there that the original article and associated material need to be stored and exported in various forms that are appropriate for teaching & learning, publication or the RAE.

As there is no need to import existing material, RAMLET use case seven (IEEE LTSC CMI Working Group, 2005) is a natural fit. This case can be summed up as produce–store–provide–retrieve, where the production of content in this case is stored directly in the central triple store, and provided in any format from the compositor component for any other system to retrieve (see Figure 8).

Advantage of such an architecture is that the export of content is optimal. Since the RAMLET ontology is a superset of all supported formats, it is also the most expressive form to author and store content in, and export with minimal loss.

Disadvantage is that the client of the storage service (not detailed in Figure 8) needs to support RAMLET RDF/OWL and accommodate its full expressivity in its business logic and its user interface. That could be non-trivial, given the nature of the RAMLET ontology.

Figure 8: Scenario 1, The Scholarly Cycle
3.1.2 The Scholarly Cycle scenario and the federated query pattern

When assuming a more distributed architectural pattern, the idea of storing content directly in RAMLET derived RDF/OWL becomes less feasible. Instead, RAMLET use case two (IEEE LTSC CMI Working Group, 2005) is more relevant. The basic pattern of this case is retrieve–interpret–disaggregate-aggregate–deploy. In the context of the federated architectural pattern, and the scholarly cycle scenario, the retrieval could be of an object authored in any one of the RAMLET supported formats from a dedicated repository directly, bypassing the compositor. The authoring tool would do the interpretation and dis-aggregation itself, and be used to edit the original article for use in a teaching and learning environment. The learning material authoring tool would also do the aggregation of the new version before deployment in a VLE (not shown in Figure 9).

This architecture would be particularly advantageous in contexts with numerous legacy systems. That includes the various repositories, but is particularly true of the VLE if it requires a proprietary content format such as in this use case.

While building a dedicated authoring tool that can take RAMLET OWL/RDF and export a proprietary content format is not as efficient as a process that takes a standard format from the compositor, the set up of Figure 9 might be more realistic and also more flexible. Such an authoring tool would also be much simpler than the one required by the
architecture outlined in 3.1.1. It would have to deal with only a small subset of RAMLET as input, and an even smaller set of semantically equivalent elements as output.

Given the right infrastructure base and a federated query architecture, coding up small, dedicated components such as the content authoring tool could be comparatively easy.

3.2 Scenario two - The all-round academic

Unlike the Scholarly Cycle scenario, the all-round academic scenario (JISC Common
Repository Interfaces Working Group, 2007) emphasises components over process. The main challenge here is the management of content across the academic's laptop and a plethora of repositories, making use of a range of services such as those provided by the Information Environment metadata registry. The transformation services provided by the RAMLET application architecture is just one of those.

3.2.1 The all-round academic scenario and the central triple store pattern

Of the various RAMLET use cases, six (IEEE LTSC CMI Working Group, 2005) is the most straightforward fit for the all-round academic scenario as it avoids re-aggregation before deployment. Use case six specifically supports the retrieve–aggregate–interpret–deploy process, where the interpret stage deals with the differences between the various complex objects just before deployment.

In the architecture proposed in Figure 11, it is assumed that the various repositories provided by the institution effectively provide the manage content process as realised by various application services. The component that realises the re-mix content service is outlined in more detail. Its function is to allow the academic to combine and re-aggregate objects, regardless of where they are from, or what format they're in. In order to homogenise the aggregated objects at deployment time, it can access both the content management functionality of the institutional repositories, and the compositor component.

The advantage of such an architecture could be that it allows instantaneous re-aggregation.
of complex objects. However, given that processing heterogeneous objects is likely to be quite challenging, and given the fact that there is a single triple store, it may seem that an authoring tool that acts directly on the normalised RAMLET RDF/OWL (as in Figure 7) is just as effective and rather simpler. You’d lose the instantaneous combination of resources that are discovered there and then, but you’d gain much quicker deployment.

3.2.2 The all-round academic scenario and the federated query pattern

Simpler still is the application of RAMLET use case three to the same situation. Use case three outlines a retrieve–interpret–store–provide process, where the retrieval is of many different formats, but the storage and provision in only one format. In the architecture outlined in Figure 12, the storage is effectively abstracted over by the compositor component while the provision of a format appropriate to the academic’s laptop is taken care of by the normalise format service realised by the compositor component.

Apart from the greater simplicity of the implementation compared to the architecture outlined in Figure 11, the solution also has the general advantage of accommodating legacy applications more easily. It does pre-suppose, however, that there will be a complex object format that is easily processed on the laptop. Given the increasing number of Atom processing tools, that might increasingly be less of an issue.

Figure 11: All-round academic scenario, central triple store pattern, manage content detail
3.3 Scenario three - Complex Objects in Teaching and Admin

Of all the CRIG scenarios, the Complex Objects in Teaching and Admin scenario is easily the most process driven as well as the most complex.

A lecturer creates a reading list for one of his MSc modules and stores it as an item on the teaching repository, which forms a basis for students' learning activity resulting in heavily annotated and amended versions of the reading list being submitted to the learning assessment repository. The examination and quality assurance processes are then managed using linked materials from repositories of teaching, teaching quality administration, and other administrative materials, with appropriate outcomes being made available via the students' eportfolio application.

(JISC Common Repository Interfaces Working Group, 2007)

To make the various processes more wieldy they have been modelled as four major, independent processes, with several subprocesses each, as illustrated in Figure 13. Note also that the lecturer actor has more than one role.
3.3.1 Complex Objects in Teaching and Admin and the central triple store pattern

Easily the most notable aspect of the scenario is the large number of repositories, which makes any implementation architecture almost necessarily a distributed one. In the centralised pattern, though, it is the triple store that is central, not the various special purpose repositories.

The most applicable of the RAMLET use cases for this situation is four, which outlines a retrieve–interpret–store–interpret–provide process. Put differently, it models the straightforward 'many formats in, store in a single format, many formats out idea'. This is operationalised in Figure 14 by envisioning a uniform Manage Content service for each repository. These services would each take care of the creation, reading, updating and deleting of content objects, or their constituent resources. The repositories are each also connected to the transform system via its various consumer interfaces, and to the compositor component via its provider interfaces.

While the architecture of Figure 14 accommodates the many legacy repositories, and has a certain conceptual simplicity to the provision of transformation capabilities, there are also numerous drawbacks. The state of the objects could easily get out of synch between the various repositories and the triple store, for example. There are also likely to be coordination and control issues between the content management services, exacerbated by the fact that multiple lossy transforms need to be guarded against.

![Figure 13: Complex Objects in Teaching and Admin scenario](image-url)
3.3.2 Complex Objects in Teaching and Admin and the federated query pattern

A rather different solution is outlined in Figure 15. This one is built around RAMLET use case 6, which deals with the possibility of multiply embedded heterogeneous complex objects. For example, a content object meant for teaching could be defined in IMS Content Packaging format, but aggregates a METS object. The process that enables such functionality can be characterised as retrieve–interpret–internalize–deploy.

In order to keep the complexity of mixing various formats manageable, implementation could be best done via SPARQL queries over RAMLET OWL/RDF representations of these complex objects, rather than their native XML. As in the architecture outlined in section 3.2.1, interpreting a heterogeneous content object at deployment time could well be too complex.

Instead, the architecture of Figure 15 builds on a service realised by the compositor component, as well as an additional access route from the compositor to the repositories. The service’s main jobs are to provide a query point for the various processes, and route objects.
This way, the command and coordination task between the processes and process stages would become much more tractable: the compositor controls. On the other hand, that does mean that the whole architecture is rather dependent on that one potential single point of failure.

### 3.4 Federated query versus the central triple store pattern

The broad comparison of each of the three scenarios in the previous sections, suggests a couple of preliminary conclusions about the application architecture patterns. Though the architectures are not always directly comparable, the clearest conclusion is the distributed pattern is more optimal in all three of the analysed scenarios.

In **3.1 Scenario one - The Scholarly Cycle**, the centralised architecture required a closely coupled, complex and custom built authoring tool, while the authoring tool required by distributed architecture could be much simpler and built from standardised components.

Likewise, in **3.2 Scenario two - The all-round academic**, the federated architecture was much simpler, and the complex object processing at deployment time less complex, and more likely to be achievable using existing object handling components.

In **3.3 Scenario three - Complex Objects in Teaching and Admin**, it was the widely distributed nature of the central pattern architecture, ironically, that led to a situation
where the transformation process could be hampered by serious control and content object state issues. The task specific, central control over a federated architecture had no such issues.

In each of these cases, there are also the generic advantages of a distributed pattern noted in chapter 2: the better accommodation of legacy systems by adding interfaces rather than replace or duplicate them, but also the greater flexibility afforded by making all data in their current state directly available. It is this flexibility that might turn out to be the most important attribute, since it is better able to accommodate new uses that have not been envisioned in any of these architectures, patterns or scenarios.

4 Implementation technology options

For either of the two architecture patterns to work, though, they have to be implementable by an institution using existing technologies. The final verdict on that judgment might well lie in an actual production implementation, but some pointers can be provided before that point.

To provide such pointers, this chapter will consider the major options for each of the data integration specific components of both implementation patterns. The options indicated are not necessarily exhaustive, but are selected on the basis of how current the software is (as indicated by the version history), and whether there is any evidence of use in current implementation projects.

4.1 Central triple store pattern options

The two main components the centralised pattern are the transform system (Figure 3) and the triple store (Figure 4). The compositor component (Figure 5) that is also part of the pattern is shared with the federated query pattern, and will be discussed separately in section 4.3.

4.1.1 Transform system

Though the transform system component handles the transition of the aggregation definition documents in XML to triples, the functionality itself doesn’t require any specific semantic technology. Rather, a series of common code libraries can be combined with some custom integration code in an application server such as Tomcat (Apache Software Foundation, 2008). The main code libraries that would be required are those that implement the interfaces: the libraries for the SWORD API (SWORD project, 2008), for
example, and the recommended ROMEO feed parser for the Atom feed interface (King, 2008). In a Tomcat environment, the obvious choice for the XSLT engine is to integrate the Xalan library (Apache XML project, 2008), but the functionality can also easily be accessed as a webservice if the implementing organisation has such a facility.

Such flexibility is even more true of the store function. At a minimum, the individual assets of a complex object can simply be exposed on a webserver. The aggregation definition triples would merely need to link to the URLs. Much more sophisticated solutions are also possible with, for example, collections in institutional repositories or the like.

One intriguing alternative to Tomcat that is used widely for semantic technology architectures is NetKernel (NetKernel Open Source Community, 2008). As a RESTful microkernel and application server, its pipeline-like means of integrating applications written in a variety of languages sits very naturally with the RESTful nature of semantic technologies (Gilman et al., 2008). Which is why it has already been used successfully in resource focussed architectures that involve XSLTs (Sletten & Wood, 2008)

### 4.1.2 Triple store

These days, there is quite a lot of choice in triple stores: the W3C semantic tools wiki currently lists 25 (Various, 2008). Some choices and distinctions can be made, though.

One aspect is whether the triple store is a hosted service or a piece of software that needs to be installed on one's own servers. The Talis platform (Talis Information Limited, 2008) is among the more mature Software as a Service offerings in this area, and provides both asset storage as well as a triple store, with SPARQL access. As such, the functionality provided covers both the asset store and the SPARQL processor outlined in the centralised triple store pattern. An inference engine should be easy to add, since the platform is essentially a hosted application server, rather than just a triple store (Davis & Ayers, 2008). Alternatively, inference engines such as the open source Pellet (Clark & Parsia LLC, 2008) or FaCT++ (Tsarkov, 2008) can be pulled in using the DIG reasoner web service protocol. The commercial AllegroGraph triple store, is also beginning to be offered as a hosted service via the Amazon Elastic Compute service (Franz inc., 2008). In that case, however, it is just the triple store functionality that is available as a service.

The other characteristic that can be used to distinguish between the various triple store options is the distinction between databases that are optimised to store triples, and programming environments or toolkits. The former focus on just storage, while the latter...
provide a number of APIs to parse, write and otherwise manipulate RDF data.

Among the dedicated triple stores, the aforementioned AllegroGraph seems widely used, while the open source Mulgara database (Mulgara Project, 2008) is moving into that space. Most noticeable is the entry of Oracle in this space with an RDF framework integrated into the Oracle 11g server product (Oracle Corporation, 2008).

The latter move also indicates an increasing trend to offer frameworks that are integrated with RDF frameworks or programming environments. The recently announced integration of Mulgara with the Sesame toolkit (Zepheira LLC, 2008) also points in that direction.

Nonetheless, many projects continue to use standalone RDF frameworks such as Sesame (Aduna, 2008), Jena (Jena project, 2008) and Redland (Becket, 2008). Of these, redland may well be the most mature (it's been developed since 2000), and has bindings for many languages. Jena has an OWL API, and a wide set of Java libraries. Sesame, however, is the only one that is continued to be backed by a commercial company. All toolkits integrate with a variety of relational databases, reasoners, and semantic development tools, though Sesame appears to have a slight edge in the breadth of its range.

In terms of the architecture, all toolkits sport SPARQL endpoints, but they all connect to external reasoners, rather than provide their own.

4.2 Federated query pattern options

The main choice in implementing the federated query pattern is whether to coincide the logical architecture with the physical architecture. The logical architecture of Figure 7 assigns an agent to each source repository, but that's not necessarily how it would be implemented. It is conceivable that all agents actually 'live' on the same application server.

Obvious advantages of doing it this way is the ability to share components, and to leave the source repositories alone as much as possible. In quite a few cases, it might well be impossible to get access to the repository's contents through anything other than the query API it happens to provide. Disadvantages of centralisation are an increase in dependencies, and a subsequent loss of flexibility. There is also more of a single point of failure.

4.2.1 Agent

The distinction between implementing the agents in a distributed or centralised fashion is not absolute, however. The inference and XSLT engines can be shared between agents even if the agents are integrated with the repositories they serve. The choices of software
package for either engine are the same as before, and largely driven by language platform considerations in the case of the XSLT, and performance in the case of the inference engine.

Other parts of the agent can be shared too. BNN's Asio (Gilman et al., 2008) and OpenLink's open source virtuoso (OpenLink Software, 2008) products both promise SQL to SPARQL bridges, or at least the tools to create them. Other than that, the RDF frameworks mentioned above can be used to construct either separate or shared SPARQL endpoints.

4.3 The compositor and other shared infrastructure

Like the transform system in section 4.1.1, the compositor doesn't require a lot of specific semantic technology. SPARQL queries can be formulated relatively easily, especially when they are based on the pre-built query templates of Figure 7.

The remainder of the compositor is not likely to be trivial because of the package aggregation functionality required, as well as the need to support a range of service providers. Fortunately, a lot of the work that is currently done to develop the conventional content transcoder (Cooper, 2008) should be re-useable for the purpose.

In addition to the collections of open and closed source products mentioned in the previous sections, there are also a number of companies that aim to provide complete semantic middleware platforms that would suit either RAMLET implementation pattern (Metatomix, 2008; Ontobroker GmbH, 2008). As these are likely to require extensive custom coding, pricing structures and other constraints could be prohibitive.

5 Recommendations

Both the range of case studies discussed in section 2.2 and the wide choice of well developed software tools discussed in 4 suggest that while standards based semantic technology is not exactly mainstream, it is sufficiently mature to start building production software with.

Combined with the persistent issue of lack of semantic interoperability and standards balkanisation, JISC could do well to investigate semantic integration technologies in practice. Several aspects could be focussed on, depending on the approximate size of the project; more than approximately 6 months and £50.000,- or less.
5.1 Small semantic integration projects
Both the federated query and the centralised triple store patterns have the compositor as the most complex custom component. With some luck, the compositor developed by the CETIS content convertor development can be re-used for that purpose. As a result, implementing both application architectures could be attainable on small budgets.

Other than observe the practical feasibility of semantic technology in general, a couple of projects could also provide baselines for either RAMLET architecture pattern, and a practical comparison. Beyond a comparison, the main choice between a centralised triple and a federated query model is one of aiming for an implementation that is near production ripeness and built on a robust stack (centralised), or aiming to investigate a potentially decisive advantage in a new and less proven set-up.

Another aspect that could be of interest to the JISC community is the comparison between implementations of components provided as SaaS and components installed on private servers. A hosted solution could open up the benefits of semantic technology for a range of institutions and staff who would otherwise not be able to participate.

5.2 Bigger semantic integration projects
Larger scale semantic integration projects could discover more about the affordances of a semantic layer technique for institutional enterprise systems. Current systems of that type may well be more complex and more time driven than typical F/HE networks, but it is equally possible that such an approach allows institutions to get value out of legacy applications, or achieve data integration that is difficult to achieve any other way.

Other aspects that can only be investigated with larger projects are the relative resource implications of using semantic technology implementations that comply with the relevant W3C standards, versus custom coding all the semantic integration components of the various architectures. Whether it is better to re-use existing standardised components, or concentrate on making cutting new code as easy as possible is an open question of some import.
6 Bibliography


Appendix 1. Complete central triple pattern RAMLET architecture, scholarly cycle scenario
Appendix 2, Complete federated query pattern RAMLET architecture, scholarly cycle scenario
Appendix 2. Complete central triple pattern RAMLET architecture, all-round academic scenario
Appendix 3. Complete federated query pattern RAMLET architecture, all-round academic scenario
Appendix 4. Complete central triple pattern RAMLET architecture, Complex Objects in Teaching and Admin scenario
Appendix 5. Complete federated query pattern RAMLET architecture, Complex Objects in Teaching and Admin scenario