Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 4, 2019

J. Algermissen
Jan Algermissen Solutions Engineering

March 03, 2019

Digital Product Life Cycle Model
draft-algermissen-digital-product-life-cycle-model-00

Abstract

This specification defines an abstract model for Digital Products and their relationships with each other in order to establish a basic abstraction on which the lifecycle of Digital Products and collaborations around them can be expressed. In addition, this specification defines a number of message formats and hypermedia controls, to enable the creation of tools and application in the space of Digital Product Life Cycle Management.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 4, 2019.

Table of Contents

1. Introduction

A wide variety of applications exists that support the journey of developers and other collaborators around the lifecycle of digital products.

The aim of this specification is to facilitate integration options between such tools and applications by providing a common abstraction and coordination protocols.

The abstract model differentiates between the notion of a Digital Product and any system or infrastructure configurations and installations created to facilitate collaboration around such digital products. A Digital Product in itself is purely abstract, primarily acting as a nexus for accountability and collaboration.

Associated with Digital Products is a life cycle model that provides an abstraction of the individual phases in which a Digital Product lives to actually produce value. Collaboration of various actors is also logically organised on the basis of these phases.

Collaboration and value creation only becomes possible if corresponding tooling or infrastructure is provided beyond the

intangible notion of a Digital Product. For each of the life cycle phases a resource abstraction is defined in order to establish a model around which software can be created for creation and management of such life cycle specific resources.

On top of this semantic foundation provided by these abstract models, this specification defines a number of message formats, hypermedia controls, and component roles that enable the creation of collaboration software systems and tools.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Glossary

Throughout this specification the following terms are used with specific meaning as defined below:

- Resource Family: An abstaction of coordinated resources that support a specific kind of usecase, such as deploying and running a product in some infrastructure or performing build, test, packaging, and publishing of products.

- Resource Family Descriptor: A file or network message containing a description of a desired instance of a resource family, for example a stage set definition file.

- Resource Manager: A software system that processes resource family descritors and creates or updates infrastructure accordingly. Examples are the creation of CI/CD pipelines and associated test systems, the creation of collaboration tool spaces and channels.

- Resource Manager Client: A software system that interacts with a resource manager.

- Provisioning Strategy: The specifics of the implementation how a given resource manager chooses to turn the abstract resource family descriptor into actually created structures. For example, a stage set resource manager could implement a strategy that puts all stage sets into a single account with shared runtimes and PaaS instances, or it could implement a strategy, where every processed stage set is provided an isolated environment.

The following resource families are define by this specification:

* Product Management Environment (TBD)

* Stage Set

* Delivery Fabric (TBD)

* Operations Cockpit (TBD)

* Report Workbench (TBD)

3. The Digital Product Life Cycle Model

This specification differentiates the following Digital Products Lifecycle aspects:

o Journey Portfolio Management Model

o Product Management Model

o Product Composition Model

o Product Content Model

o Product Delivery Model

o Product Deployment Model

o Product Operations Model

4. Services

The Digital Product Life Cycle Model aims to facilitate the integration into existing Internet technology and thus uses existing specified semantics whenever possible.

One of such integration points is the notion of _Service_, which refers to an abstract capability associated with an interaction protocol expectation. This specification uses the term _Service_ in the exact same sense as it is used by the various specifications available through the IETF.

Some relevant specifications are [RFC3232]. [RFC2782], [RFC6763].

On the one hand services are used by capability providing software systems to advertise that they meet the contract associated with a given service name, or in other words, that the providing system

meets the expectations of consuming systems of a given service. On the other hand, services are used by consuming systems to express that they have certain protocol and capability expectations when interacting with a providing system.

While services in most existing specifications are rather protocol focussed, the service notion seamlessly supports referring to capabilities that exist in more functional areas. The following examples show services that illustrate the notion of _servciew_ as understood by this specification:

'search', 'suggest', 'basket', 'ftp', 'postgres', 'smtp', 'http', 'aws-dynamo', 'gcp-pubsub', 'google-maps',...

4.1. Service Names

Service names MUST conform to the syntax requirements stated in [RFC1034], meaning that service names MUST only consist of ASCII letters, digits, and hyphens and that they MUST NOT be longer than 15 characters.

Service names are case insensitive.

TBD: Differentiate between standardized global services and context-based services

4.2. Port Numbers

Services MAY be assigned a port number, see [RFC3232] and https://www.iana.org/assignments/port-numbers

4.3. Service Catalog

In order to express dependencies of a product on one or more services, the list of services available in a given context must be known. Systems that have the ability to manage dependencies SHOULD expose service catalogs that list the available services.

4.4. The service-catalog Link Relation Type

Links with the link relation type 'service-catalog' indicate that the target resource represents a service catalog.

4.5. Service List Documents

The canonical model for a service list document is a JSON [RFC8259] object.

When serialized as a JSON document, that format is identified with the "application/vnd.ply.servicelist+json" media type.

4.5.1. Syntax Example

```
{
  "services" : [
    {
      "name" : "",
      "description" : "...",
      "docs" : [ "" , "" ]
    },
    {
      "name" : "",
      "description" : "...",
      "docs" : [ "" , "" ]
    }
 ]
}
```

5. Execution Environment Types

Common to all software systems is the differentiation between something being developed and actually running it in a given context. The things being develop are inherently bound to the target type of execution environment. This specification defines the notion of _Execution Environment Type_ in order to capture this property of developed software items. An Execution Environment is anyting into which a developed software can be deployed to realize its capabilities in a given runtime environment.

Examples of Execution Environment Types are the usual environments such as "Linux Operating System", "Docker", "AWS Lambda", "Java Application Server", "Oracle PL/SQL", but other possible runnable artefacts and Execution Environment Types are "Single Page Applications deployed to a CDN", "The configuration of an integration proxy", "A native mobile app deployed on a mobile phone".

5.1. Execution Environment Type Definitions

Execution Environment Types establish a contract between product component developers and processors of component deployments and developers need to understand this contract when they develop the component. For example, part of this contract is the definition of how the well known environment variables are provided to the component at runtime. Another part is how component artifacts must be published before or during deployment.

The contract established by a given Execution Environment Type must be made availble to the users as Execution Environment Type Definitions and must convey the foplowing information:

5.1.1. Name

The identifier and name of the execution environment type.

5.1.2. Description

A detailed description of the contract in a way that is sufficient for the user to develop components and publish them. Specifically this MUST include

- What is the expected artifact build and packaging format?
- What is the expected runtime behaviour (TBD: explain)
- What are the startup and shutdown constraints
- What are the deployment parameters that MUST, SHOULD, or MAY be passed as party of deployment (eg cmount CPU)
- How are the environment variables passed to the component at runtime?
- How are secrets pertaining to individual variable values provided?
- Anything else the developer needs to know the execution environment will expect from her software.

5.2. Execution Environment Variables

TBD

5.3. Execution Environment Type Catalog

Systems that support the deployment of software artifacts SHOULD provide an execution evironment type catalog to inform client systems about the supported execution environments.

5.4. The 'execenv-catalog' Link Relation Type

Links with the link relation type 'execenv-catalog' indicate that the target resource represents an execution environment type catalog.

5.5. Execution Environment Type List Documents

TBD

The canonical model for an execution environment document is a JSON [RFC8259] object.

When serialized as a JSON document, that format is identified with the "application/vnd.ply.execenvlist+json" media type.

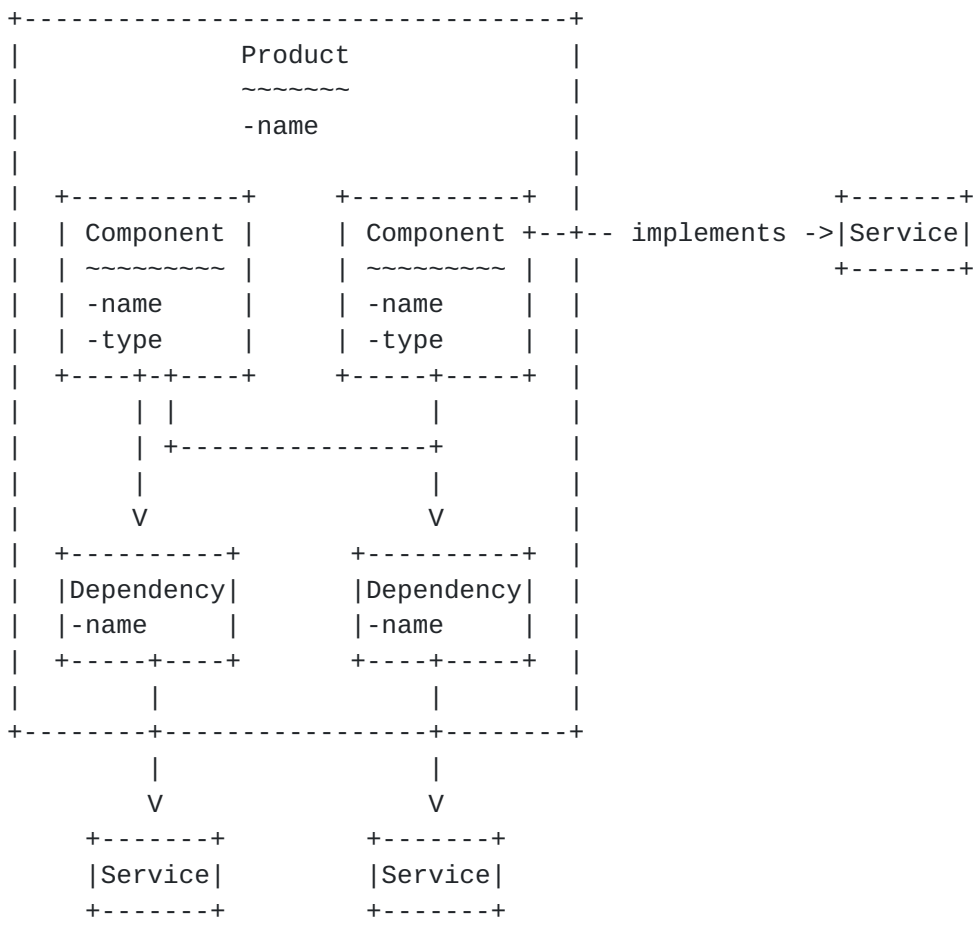
5.5.1. Syntax Example

```
{
  "execenvs" : [
    {
      "name" : "",
      "description" : "...",
      "docs" : [ "" , "" ] },
    {
      "name" : "",
      "description" : "...",
      "docs" : [ "" , "" ]
    }
 ]
}
```

6. Digital Product Composition Model

A digital product is understood to consist of components. Components can have dependencies on services. Services are either backing systems like databases or they are external services, such as a search API, a backend IT system, or another product in the sense of 'product' used in this specification.

Components can share dependencies that are specific to a product, but no two products may share the same service instances.



6.1. Product

TBD

6.1.1. Product Name Attribute

Product objects MUST exhibit a "name" property with a string literal value which conforms to the syntax requirements for DNS labels. The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less ([RFC2181]).

6.2. Component

6.2.1. Component Name Attribute

Component objects MUST exhibit a "name" property with a string literal value which conforms to the syntax requirements for DNS labels. The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less ([RFC2181]).

6.2.2. Component Type Attribute

Components have an Execution Environment Type indicated by the 'type' attribute.

6.3. Dependency

TBD

Dependencies are named to differentiate between them. This makes it possible for a product to exhibit one or more dependencies on the same service.

6.4. Service Implementation

When products expose capabilities for use by other products, they must expose them in order for other products to consume them as dependencies.

7. Product Documents

In order to communicate product and product structure information between systems, this specification defines a syntax for product documents. In addition to the structural product information, product documents contain syntax elements that enable the coordination between resource managers and resource manager clients.

Resource managers SHOULD use these syntax elements to embed discovery information into product data to enable resource manager clients to determine where to send product data- or resource descriptor updates and where to retrieve update status information.

Resource manager clients SHOULD leverage the discovery elements as much as possible and understand product documents as the primary means of coordination between resource managers and clients.

Focussing resource manager client development on the semantics of product documents and other hypermedia elements defined in this

specification enables the creation of generic user agents for any resource manager implementation. In other words, resource manager clients SHOULD not rely on specific aspects of a certain resource manager instance to avoid coupling the client to the original design of that specific manager.

7.1. The Product JSON Object

The canonical model for a product document is a JSON [RFC8259] object.

When serialized as a JSON document, that format is identified with the "application/vnd.ply.product+json" media type.

7.2. A YAML-formatted example of a product model

```
name: search
dependencies:
  - name: index
    service: solr
  - name: config
    service: postgres
components:
  - name: importer
    dependencies: ["index","config"]
  - name: searcher
    dependencies: ["index"]
    implements: ["opensearch","my-suggest"]
```

8. Stage Sets

Stage Sets are a resource family that enables the creation of sets of related stages in a given infrastructure providing system. Resource Managers that accepts the processing of stage sets, are expected to create and configure infrastructure accoring to the desired target state as abstractly expressed in the processed state set.

Many stage sets can be defined for any given product and different resource managers can be chosen to be responsible for any number of such stage sets.

For example, one might choose (1) to apply one resource manager to a public cloud provider for managing the main delivery stages there, (2) to apply another resource manager to an on-premise infrastructure and manage individual, short lived, per-feature branch test stages in

a cheaper environment, and (3), developers could use a local resource manager to manage local development stages and generate Docker Compose-based running systems from them.

Therefore, stage sets enable a fully decentralized management of sets of related stages with, propably independent, collaborators or software systems.

By allowing unrelated resource managers to manage unrelated sets of stages the approach also provides a clean path for migrating from one resource manager to another, for example, when switching infrastructure providers.

8.1. A Note on Provider-Coupling

In order for resource managers to instantiate the necessary infrastructure (for example network) and PaaS-level structures (for example databases), and in order to perform component artifact deployment, infrastructure provider specific services and their configuration parameters need to be part of stage set definitions. The notion of stage sets does not aim to abstract from the specifcs of the chosen target infrastructure beyond maybe reducing its complexity by providing useful named predefined configurations (aka "T-shirt sizes").

9. The Stage Set JSON Object

The canonical model for a Stage Set document is a JSON [RFC8259] object.

When serialized as a JSON document, that format is identified with the "application/vnd.ply.stageset+json" media type.

9.1. A YAML-formatted example of a stage set model

```
name: myset
'product-ref': ply://acme/search
budget: '66474-gghk-88733/00'
'status-href': http://example.org/acme/search/stagesets/status
stages:
  - name: dev
    criticality: work
    dependencyInstances:
      - name: mydb
        parameters:
          - name: foo
            value: bar
    componentDeployments:
      - name: searcher
        artifact: myrepo.example.com/images/foo:1
        dns: www.example.org
  - name: prod
  ...
```

10. Delivery Fabrics

TBD: Describe delivery fabrics and their documents analog to stage sets.

11. Extension Resource Families

This specifications defines two resource families, stage sets and delivery fabrics.

New resource families can be defined outside of this specification as extensions. The following SHOULD be supported by such extensions:

TBD: What can be required from extensions to allow for some generic tools support?

12. Well Defined Execution Environment Variables

TBD Define the list of common environment variables for execution environments

- PLY_LOC_STAGE
- PLY_LOC_REGION
- PLY_LOC_SYSTEM
- ...

13. Example Implementations

TBD

14. Security Considerations

TBD

15. IANA Considerations

This specification defines new Internet media types [RFC6838].

15.1. application/vnd.ply.product+json

- Type name: application
- Subtype name: vnd.ply.product+json
- Required parameters: None
- Optional parameters: None; unrecognized parameters should be ignored
- Encoding considerations: Same as [RFC8259]
- Security considerations: see Section 5 of this document
- Interoperability considerations: None
- Published specification: TBD (this document)
- Applications that use this media type: HTTP
- Fragment identifier considerations: Same as for application/json ([RFC8259])
- Additional information:
- Deprecated alias names for this type: n/a
  - Magic number(s): n/a
  - File extension(s): n/a
  - Macintosh file type code(s): n/a
- Person and email address to contact for further information: Jan Algermissen algermissen@acm.org [1]

- Intended usage: COMMON
- Restrictions on usage: None.
- Author: Jan Algermissen algermissen@acm.org [2]
- Change controller: IESG

15.2. application/vnd.ply.product+yaml

TBD

15.3. application/vnd.ply.servicelist+json

TBD

15.4. application/vnd.ply.servicelist+yaml

TBD

15.5. application/vnd.ply.execenvlist+json

TBD

15.6. application/vnd.ply.execenvlist+yaml

TBD

15.7. application/vnd.ply.stageset+json

TBD

15.8. application/vnd.ply.stageset+yaml

TBD

15.9. The ply URI scheme

TBD

15.10. The service-catalog link relation

TBD

15.11. The execenv-catalog link relation

16. References

16.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC3232] Reynolds, J., Ed., "Assigned Numbers: RFC 1700 is Replaced by an On-line Database", RFC 3232, DOI 10.17487/RFC3232, January 2002, <https://www.rfc-editor.org/info/rfc3232>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

16.2. Informative References

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

16.3. URIs

[1] mailto:algermissen@acm.org

[2] mailto:algermissen@acm.org

Appendix A. Acknowledgements

Thanks to TBD for their comments.

Appendix B. Implementation Notes

TBD

Author's Address

Jan Algermissen
Jan Algermissen Solutions Engineering

EMail: algermissen@acm.org
URI: http://algermissen.io