Context Map

Overview

This page describes the high-level dependencies between the various contexts

The solution domain comprises several distinct Contexts that collaborate to provide an end-to-end solution to the problem at hand.

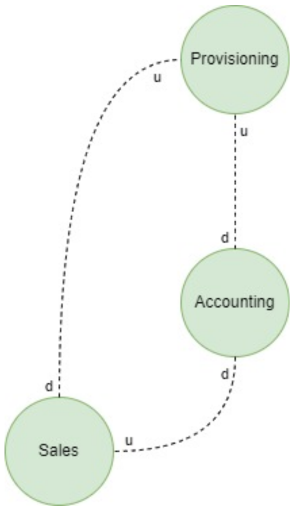
A **Context Map** identifies the Contexts in a solution domain as well as the dependencies that exist between each Context.

The relationships between each Context convey a *knowledge* dependency and not a *runtime* dependency. A relationship between two Contexts is defined with a line and contains annotations *u* and *d*. They have the following meanings:

- u - upstream dependency
- d - downstream dependency

The Contexts are also visually organized so that those with an upstream dependency are shown above the downstream Context.

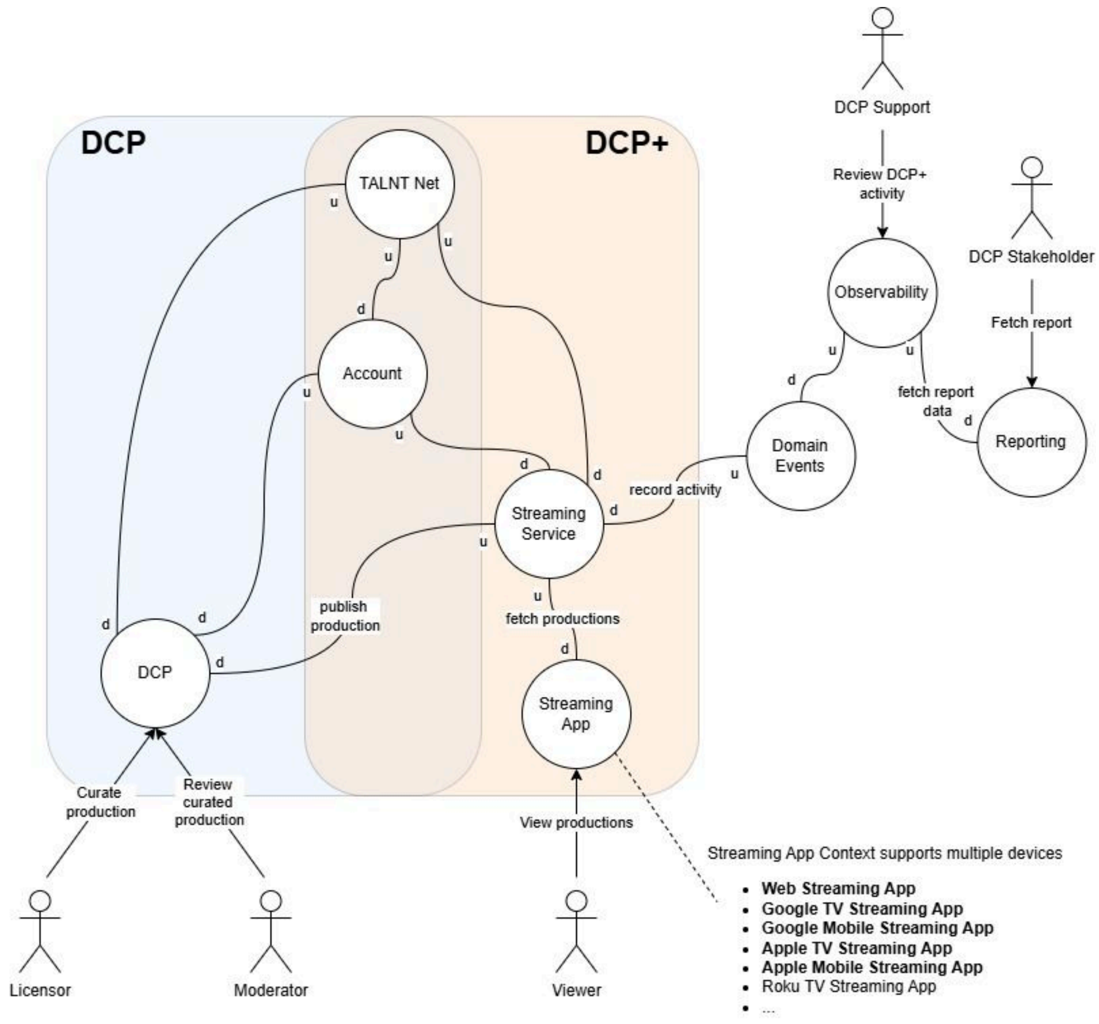
A Context Map conveys that a downstream Context has a knowledge dependency on the upstream Context. Generally speaking, the downstream Context *depends* on information from the upstream Context in order to facilitate its set of responsibilities.



For example, consider an analogy of a set of departments within an organization. This Context Map conveys the dependency between Provisioning, Sales, and Accounting. Sales depend on both Provisioning and Accounting.

- Sales informs Provisioning to provision a new customer with services that were just sold to the customer.
- Sales informs Accounting that there is a new customer with services that will require billing.
- Accounting depends on Provisioning to know service usage to provide usage-based billing.

Context map



The following diagram conveys the knowledge dependencies between the various contexts in our solution domain.

Context	Comments
TALNT Net Context	This context is for TALNT Net that operates on the Base L2 and AppChain. It is responsible for executing Smart Contracts related to DCP and DCP+ and holding state that must be secure, immutable and transparent.
Account Context	This is the top-most context responsible for managing user accounts that have access to Decentralized Pictures applications such as the DCP and DCP+ It is responsible for handling user data and activities common to both DCP and DCP+, while allowing user data and activities specific to DCP and DCP+ to be siloed in the application that concerns them.
DCP Context	This is the primary context for the DCP application. In addition, it supports the production curation features required by the DCP+ application.

Streaming Service Context	This context is for the DCP+ application. It is responsible for accepting curated content from the DCP Context and making that content available to the Streaming App context. It also centrally tracks state required by the Streaming App, allowing all instances of the Streaming App to rebuild their state from scratch and share state across all supported device types. The Streaming Service Context handles the heavy lifting in determining what content to provide to a Streaming App, thereby simplifying the development efforts of the Streaming App, enhancing consistency across Streaming Apps, and minimizing Streaming App processing and storage requirements to prevent overloading low-performing devices, such as TVs.
Streaming App Context	This context is for the DCP+ application. It is responsible for fetching content from the Streaming Service and making it available to viewers. The Streaming App context supports multiple devices, including web browsers, Google Smart TVS, and Google mobile devices. It can easily be extended to support other devices, such as Roku smart TVs, Apple TVs, and Apple mobile devices.
Domain Event Context	An entity tracks its state and the rules regulating its lifecycle. It also tracks the actual cause of the state changes, allowing one to explain how the system evolved into its current state. A distinct, though related set of issues arises in distributed systems. A distributed system's state cannot always be kept completely consistent. Within a Context, the aggregates are always internally consistent while making other changes asynchronously across Contexts. A record of state and life-cycle changes is encoded in a domain event, also referred to as an event. An event captures the memory of a change made to an entity. It is a fully fledged part of the domain, representing something that has occurred within the domain. Each executed *task* supported by our product will be recorded as an event and transmitted to downstream Contexts that depend on that event, as well as to an observability system where users can monitor product activity. An event is fact-based in that it describes what happened, what triggered it, what was affected, when it occurred, and who was responsible. An event does not seek to provide an interpretation as to why the activity occurred. Interpretation happens with knowledge workers and interpretation systems that analyze events using higher-level analysis. Domain events: • is the primary means for company employees to understand system behaviour and usage so we can: ◦ help identify where and how we should best invest our resources to enhance and optimize our product. ◦ understand unusual changes in customer usage so we can proactively reach out to them and determine if they need some assistance, ◦ to know how technical resources are being utilized or project technical resource scaling demands so we can better manage our cloud infrastructure and minimize costs. • is a key requirement to facilitate operational intelligence. It is crucial to have timely access to events to optimize our ability to operate our product with the fewest personnel in operational roles. Domain Event logging enables customer support, root cause analysis, outage detection, defect detection, feature adoption, customer attrition, security breaches, etc. Customer-facing staffers will have near-real-time and direct access to the data they need to perform their role. • are the primary means to facilitate loosely coupled intra-component collaboration using an event-driven architecture pattern
Observability	
Reporting	