ネットワークシステムデザインにおける
エキスパート思考パターン

～ 2016/1/21 JANOG37 BoF 用資料 ～

2016/1/21
佐々木 健

エキスパートの方々にネットワークシステムをデザインする際に考慮していること等のヒアリングを行ない、その内容をデザインパターン形式にまとめた資料です。

基本的な考え方

No.1 シンプルに作る
No.2 汎用的な技術を使う
No.3 充分なリソースを用意する

要件を満たす

No.4 目的に合った実装
No.5 技術以外の課題も考慮する

考慮すべき技術要素

No.6 ボトルネックポイント(制約条件)を知る
No.7 優れたアイディアを知る

哲学

No.8 自律分散
No.9 ルール・イン・ザ・パケット(ステートレス)
No.10 デクレラティブ(処理ではなく性質を定義する)

Special Thanks to
Miya Kohno,
Ryuichi Takashima,
Satoru Matsushima.

No.1

Pattern Name: シンプルに作る

Context: システム設計、プロトコル設計時

▼その状況において
Problem: 使わない機能ができたり、同じような複数の機能ができたりしてしまう。それにより実際には使いにくくなったり、発展させられなくなったりしてしまう。

（なぜこのような問題が生じるのかと言うと）
Forces:
・いろいろな機能を一つに組みこもうとしてしまう
・必要な機能の吟味が足りない

▼そこで
Solution: 必要充分な機能をシンプルに作るように心がける

（具体的な例）
Samples:
・イーサネット
・データプレーンラーニング
・リングトポロジ
・鉄道で使われている磁石電話

▼その結果
Consequence:
（Problem-Solved）わかりやすくなるとユーザー数が増える。障害対応もしやすくなる。拡張も容易になる。

（Additional）

No.2

Pattern Name：汎用的な技術を使う

Context：システム設計における技術選択時

▼その状況において
Problem：新しくて流行っている技術を適用することにより、問題が発生することがある。

（なぜこのような問題が生じるのかと言うと）
Forces：
・新しい技術を使ってみたいと思うのはエンジニアの性
・新しいものは良いものだ、という思いこみがある

▼そこで
Solution：枯れていてちゃんと動く技術をベースにすることを検討する

（具体的な例）
Samples：
・BGP、OSPF は使い回しが効く。枯れていてちゃんと動く。わりとどこでも使える。
・BGP はコアロジックがシンプルで拡張性もある。
・IPv6 はメタデータを運び放題で便利。
・ルートリフレクターはコントローラとしても使えたり汎用的に使える

▼その結果
Consequence：
（Problem-Solved）枯れた技術の適用部分に安定感を得ることができる。

（Additional）

No.3

Pattern Name：充分なリソースを用意する

Context：システム設計時

▼その状況において
Problem：ネットワーク帯域等のリソースをケチると結果的に損をするケースがある

（なぜこのような問題が生じるのかと言うと）
Forces：
・節約するために考えるポイントが増えてしまう。
・節約のための仕組みの分複雑さが増えて、その分のコストも増えてしまう。
・リソースが足りなくなってしまった際の解決は難しい。

▼そこで
Solution：ケチケチせずに充分なリソースを確保しておく。

（具体的な例）
Samples：
・キャリアバックボーン
・ストレージネットワーク

▼その結果
Consequence：
（Problem-Solved）システムの安定と安心を手に入れることができる。

（Additional）リソースを過剰に用意する、ということではない。

No.4

Pattern Name：目的に合った実装

Context：システム設計時、実装時

▼その状況において
Problem：本来の目的に合わない技術を選択してしまう

（なぜこのような問題が生じるのかと言うと）
Forces：
・目的があいまいな状態で設計をしてしまう
・ついつい流行りの技術を使いたくなる
・技術選定が甘い

▼そこで
Solution：目的を定義し、目的達成できる技術選定を心掛ける。

（具体的な例）
Samples：
・キャリアの MPLS バックボーン(どんなトラフィックも吸収でき思ったところに通せる)
・TRILL(L2 をわりと簡単に作れる)

▼その結果
Consequence：
（Problem-Solved）システムオーナー、利用者に対して胸をはれる。

（Additional）弱点についての説明は必要
・MPLS は運用者にスキルが必要になる
・TRILL は障害ドメインが広くなるので使いどころを考える必要はある

No.5

Pattern Name: 技術以外の課題も考慮する

Context: システム設計時、構築時、運用中

▼その状況において
Problem: 技術に直接関係がないところが運用上の制約になることもある。それにより本来やりたいことが実現できないこともある。

（なぜこのような問題が生じるのかと言うと）
Forces:
・運用者や利用者に様々な好みがある。
・歴史的事情
・人間関係に制約がある

▼そこで
Solution: 技術以外の外的要因もシステム構築時、運用時に考慮する。

（具体的な例）
Samples:
・BGP のピアリング交渉
・トラフィックエンジニアリング

▼その結果
Consequence:
（Problem-Solved）本来やりたいことが円滑にできるようになる。

（Additional）外的要因の影響を受けにくいシステムをうまく作れれば回避はできる。

No.6

Pattern Name: ボトルネックポイント(制約条件)を知る

Context: 大規模システム構築、拡張時

▼その状況において
Problem: システム規模に合わないプロトコルや実装を選択してしまうことがある。

（なぜこのような問題が生じるのかと言うと）
Forces:

・慣れた技術を安易に適用してしまうことがある。
・そもそも小さい規模での適用のみを想定している技術もある。
・大規模システムに適したトポロジーを想定していない技術もある。
・本質的にスケールしにくい仕組みになっている技術がある。
・仕様策定時に想定されてなかったことが、今は要求されることがある。

▼そこで
Solution: スケールしにくい技術を回避し別の適切な技術を選択する。スケールするように実装方法を工夫する。要求に応えられるように技術を拡張する。

（具体的な例）
Samples:

・イーサネットネットのブロードキャスト
・MPLS のコントロールプレーン
・ATM のリソース空間の制限
・フレームリレーでは多段構成が持てない
・3GPP では Point to Point のトポロジしか取ることができない

▼その結果
Consequence:
（Problem-Solved）システム拡張が可能になり、システムの寿命も伸びる。

（Additional）他の弱点についても把握することでより適切な技術選択もできる。

No.7

Pattern Name：優れたアイディアを知る

Context：技術を評価しているとき

▼その状況において
Problem：流行っていない実装の中のキラリと光るアイディアを見逃してしまう

（なぜこのような問題が生じるのかと言うと）
Forces：
・流行っていない技術はそもそも評価対象にならない
・具体的なユースケースだけを考えてしまう

▼そこで
Solution：技術のアイディア的なところに目を付けるようにする

（具体的な例）
Samples：
・Diffserv aware TE(クラス分けという概念は良かった)

▼その結果
Consequence：
（Problem-Solved）他のシステムを作る際にアイディアを取りこむことができる

（Additional）

No.8

Pattern Name：自律分散

Context：プロトコル設計、システム設計時

▼その状況において
Problem：集中管理をする部分があるとそこが弱点になってしまう。

（なぜこのような問題が生じるのかと言うと）
Forces：
・集中管理ができるとシステムの見通しが良くなる
・管理するものが増えると管理や運用がたいへんになってくる。
・集中管理しているところが壊れるとシステム全体がダウンしてしまう

▼そこで
Solution：個々が自律的に動けるような分散システムの仕組みを考える

（具体的な例）
Samples：
・BGP
・DNS

▼その結果
Consequence：
（Problem-Solved）一部が壊れても全体への影響を最小限にできるようになる

（Additional）分散システム固有の難しさはある

No.9

Pattern Name: ルール・イン・ザ・パケット(ステートレス)

Context: プロトコル設計、システム設計時

▼その状況において
Problem: ステートを管理する部分がボトルネックや高コスト要因になってしまう

（なぜこのような問題が生じるのかと言うと）
Forces:
・ステートを管理する仕組みのほうが考えやすい
・ステート管理が必要だという思いこみも発生しやすい
・ステート管理部分はシングルポイントになりがち。冗長化も難しい。

▼そこで
Solution: ステートレスにできないか、ルールをパケットに入れられないか熟慮する

（具体的な例）
Samples:
・6rd、4rd
・MPLS
・セグメントルーティング

▼その結果
Consequence:
（Problem-Solved）ボトルネックや高コスト要因をなくすことができる

（Additional）ステート管理がやはり必要なケースはもちろんある。

No.10

Pattern Name： デクレラティブ(処理ではなく性質を定義する)

Context： プロトコル設計、システム設計時

▼その状況において
Problem： 大規模システムで処理を定義をしていくとその数が増えすぎて管理不能になる

（なぜこのような問題が生じるのかと言うと）
Forces：
・処理を定義していくのは直感的にはわかりやすい
・システムが大きくなると１つのやりたいことに対する処理定義の量も増える
・量が多くなると処理と処理の連携がわかりにくくなってしまう

▼そこで
Solution：処理を定義するのではなく性質を定義するようなパラダイムを導入する

（具体的な例）
Samples：
・Group Based Policy
・CISCO ACI

▼その結果
Consequence：
（Problem-Solved）大規模システムでの管理を楽にできるようになる

（Additional）宣言型プログラミングというパラダイムからのアイディア。