システムの変化に追従可能でかつ理解し易いドキュメントシステムのモデル化

1. モチベーション

- システムの運用をするためにはシステムの理解が必要
- ところが、システムは継続的に開発されるものであり変化するものである。また、その変化のスピードは早くなっている(要出典)
- また、システムは巨大で複雑なものになっており1人で全てを理解するのは現実的に難しくなってきている
- 上記のような状況の中、継続的にシステムを運用していくためのシステム理解の効率的な手法を得たい
    - ゼロからシステムを理解する場合
    - 変化したシステムについて理解の追従をさせる場合

2. ターゲット

- 「既にユーザに機能を提供しており運用が必要なシステム」にターゲットを絞る。新規開発などの設計段階のシステムは含まない。
- システムを理解する手法としてドキュメントに注目する

3. システムとそれを説明するドキュメントのモデル化

前提

- エンジニアがシステムを理解しようとするとき、ドキュメントの内容が正しくシステムを説明している場合、システムを直接見て理解するより、ドキュメントを通じて理解するほうが理解し易いとする
    - つまり、本稿は「いかに正しいドキュメントを得るか」を主題とする

システム

- あるシステムにおいて、理解しなければならない要素を本稿では「システム要素」と呼ぶ。システム要素を$s_1, s_2, \cdots s_n$とし、それらを元とした集合 $s_1, s_2, \cdots s_n$ をSとする
    - システム要素は、例えばインフラであればサーバ、コンテナ、ミドルウェア、ネットワークなど

ドキュメント

- システムを説明するドキュメントを集合Dとする。ドキュメントはシステム要素を説明するものである。
    - 「ドキュメントDがあるシステム要素s_nを説明できている」場合、関数$desc()$をつかって$desc(s_n) \in D$とする

（関数とは集合Aの各要素に集合Bの唯一つの要素を割り当てるもの）

真に正しいドキュメント

$$desc : S \to D \tag{1}$$

また

$$desc(S) = D \tag{2}$$

（外延性の公理より、集合としては元の重複は考えない）

正しいことが記載されているが不十分なドキュメント

$$D \subseteq desc(S) \tag{3}$$

（Dは$desc(S)$の部分集合）

ある集合Aの要素の個数を記号$n(A)$で表すとする。

$n(D \cap desc(S))$が$n(S)$に近ければ近いほど良い。

$n(S) - n(D \cap desc(S)) = 0$ が理想。

間違ったことが記載されているドキュメント

$desc(s_x) \in D$かつ$s_x \notin S$となる元s_xが存在する

これを本稿では「間違い要素」と呼ぶ。間違い要素の集合は、差集合$D - desc(S)$ で表すことができる。

時間の経過に応じて変化するシステムとドキュメント

システムは継続的に開発されるものであり変化するものである。

ある時刻tのシステムを S_tと表す。

時刻tが、$t_1, t_2, \dots t_n$と経過する場合、それぞれのシステムを$S_{t_1}, S_{t_2}, \dots S_{t_n}$とすると、

簡易的に、システムの巻き戻し（revertなど）を考えないとし

$$n(S_{t_1} \cap S_{t_1}) = n(S_{t_1}) \tag{4}$$

$$n(S_{t_1} \cap S_{t_2}) > n(S_{t_1} \cap S_{t_3}) > \dots > n(S_{t_1} \cap S_{t_n}) \tag{5}$$

と、システム要素の時間の経過に応じて変化していくとする。

この場合、ドキュメントDが仮に$D = desc(S_{t_1})$だとしても、時間の経過と共に$n(D \cap desc(S_{t_n}))$の数は小さくなる。

これは「正しく説明できているシステム要素」の数が時間の経過に応じて小さくなることを表している。

また間違い要素集合も$n(D - desc(S_{t_n})) > 0$として存在することになる。

システムの変化に追従するドキュメント

これはSのシステム要素の変化に応じてDの要素が追従できれば良い。

人が実施するのであれば「システムの変化に応じてドキュメントを書き直せばよい」ということになる。

ただ、近年システムの変化のスピードは早くなっている。またシステムは巨大で複雑なものになっており、手動でシステムに変化に追従するのは難しい。

どこでドキュメントの自動生成が考えられる。

まず、ドキュメントDを生成する関数を$f()$とした場合、時刻によって変化するS自体を入力とした$f(S)$を作るということがドキュメントの自動生成を表すと言える

$$D = f(S_{t_n}) \tag{6}$$

さらにその結果が $desc(S)$ となれば、Sの変化に追従できるといえる。

$$D = f(S_{t_n}) = desc(S_{t_n}) \tag{7}$$

つまり「システムの情報からドキュメントを生成する」というアプローチが良さそうということがわかる。

システムとドキュメントを変化に応じて生成する

今までのモデルで別のアプローチも表現できる。

あるシステムSは常に設定Cから生成されるもの($gen()$)であった場合、

$$gen(C_{t_n}) = S_{t_n} \tag{8}$$

また設定Cからドキュメントも生成できれば($doc()$)、

$$doc(C_{t_n}) = D \tag{9}$$

$$C_{t_n} = doc^{-1}(D) \tag{10}$$

$$D = doc(C_{t_n}) = doc(gen^{-1}(S_{t_n})) \tag{11}$$

となり（逆関数があるとする）

$doc(gen^{-1}())$ が $f()$ といえる。

OpenAPIなどがこのアプローチにあたる

4. システムの理解のしやすさのモデル化

セクション3では、前提に「ドキュメントを通じてシステムを理解する」と置いている。

また、真に正しいドキュメントとして $D = desc(S)$ とモデル化している。

つまりシステム要素の数とドキュメントの要素の数は同数となる

$$n(D) = n(S) \tag{12}$$

近年システムは巨大で複雑なものになっており1人で全てを理解するのは現実的に難しくなってきている。

つまり、人一人が最終的に理解できる要素の数をmとすると

$$m < n(S) \tag{13}$$

であるということになる。

また、人一人が一度に理解できる要素の数をkとすると

$$k < m < n(S) \tag{14}$$

となる。

ドキュメントDを理解しやすくするためには、部分集合$D_1, D_2, D_3 \dots D_n$に分割してそのそれぞれのD_nは

$$k < n(D_n) \tag{15}$$

であることが望ましいことがわかる。

部分集合の分け方については、さらに理解しやすくするために様々なアプローチが取れる。

例えばサーバ一覧やネットワーク図など、同じようなシステム要素をまとめてグルーピング化しても良いし、各マイクロサービスごとに分けても、ある処理ごとに分けても良い。

これを ISO/IEC/IEEE 42010:2011 にならって「ビュー」と呼ぶ（ただし本家のそれとは厳密には定義が異なる）。

ビューはステークホルダー（そのドキュメントの対象者）が理解すべきコンテキストに沿って記述される。

ネットワーク管理者にとってはネットワーク図、あるマイクロサービスを管理するSREにとってはそのマイクロサービスを構成するノードとネットワーク、など。

よってD_1に含まれる要素がD_2に含まれることもある。

よって理解しやすさを持ったドキュメントの特徴としては以下のようになる

$$D = D_1 \cup D_2 \cup D_3 \cup \dots \cup D_n \tag{16}$$

5. ドキュメントシステムに必要な要件

「継続的にシステムの変化に追従できる」「理解しやすい」ドキュメントシステムの要件として

セクション3より、

$$D = f(S_{t_n}) = desc(S_{t_n}) \tag{17}$$

であること、セクション4より、

$$D = D_1 \cup D_2 \cup D_3 \cup \dots \cup D_n \tag{18}$$

というドキュメントが生成できることと主張した。

ここで、さらに$f(S)$について$desc(S)$とするために分解すると、

1. 入力Sからシステム要素を列挙すること
2. 1で列挙したそれぞれのシステム要素について理解のための文章を追加すること（本稿ではアノテーションと呼ぶ）

と分解できそうである。

ここで、2のアノテーションは技術的に困難であるとして人に任せるとすると、1だけが要件として残る。

つまり、

1. 対象システムからシステム要素を抽出でき、それぞれに対して穴埋め形式でドキュメントを記述できる
    - もしくは$f^{-1}(D) = S$を検証できる仕組みを持つ
2. 1で抽出したシステム要素を任意のグループでグルーピングすることができ、それに対してドキュメントを記述できる
    - システム要素はグループ間で重複可
    - 1で抽出した要素のみで2を実現することで$s \notin S$となる要素が存在しないように

という要件が見えてくる。

ndiag

上記の要件を満たすドキュメンテーションツールとして筆者はndiagというツールを作成している

https://github.com/k1LoW/ndiag

ndiagはシステム要素の抽出機能を持っていないので、例えばノード間のリレーションなどはサービスマップや https://github.com/yuuki/shawk などを入力とするようなことを想定している

参考資料

- 目的に沿ったDocumentation as Codeをいかにして実現していくか / PHPerKaigi 2021
- ソフトウェアシステムアーキテクチャ構築の原理 第2版