SpaceFox

Zeste de savoir

Dates, durées et horloges en informatique

7 mars 2022

Table des matières

Introduction

Le concept de «temps» a ceci de paradoxal qu'il est à la fois très intuitif (tout le monde sait ce qu'est *le temps*) et extrêmement complexe dans ses implications et ses représentations[1]footnote:1.

C'est particulièrement vrai en informatique, où l'on a beaucoup d'outils pour manier les informations temporelles dans toute leur subtilité. Mais ces moyens sont souvent mal utilisés, soit parce que mal compris, soit parce que *les notions qu'ils font intervenir* sont mal comprises, parfois par les concepteurs des langages ou des bibliothèques. Or, une méconnaissance, un emploi du mauvais outil peuvent mener à des erreurs complexes, rares, délicates à trouver et à corriger, et aux conséquences démesurées – imaginez un système de paie qui ne se lance pas, par exemple.

Le but de ce tutoriel est de faire le point sur la gestion de la temporalité en informatique: réviser et clarifier les concepts de **date**, **durée** et **horloge**, et voir comment manipuler ces concepts en informatique.

i Prérequis

Ce contenu n'a aucun prérequis particulier. Même si vous ne programmez pas, il pourrait vous servir pour votre culture générale.
Les exemples seront en Java «récent» (8 ou plus), mais les concepts sont applicables à tous les langages. Les différents outils utilisables pour chaque langage seront regroupés en annexe. N'hésitez pas à indiquer ceux qui concernent votre langage préféré en commentaire, que je puisse le rajouter à la liste!

Les trois parties principales devraient se lire dans l'ordre proposé, car chacune part du principe que la précédente est acquise.

1. [2]footnote:1 Par exemple, il n'existe pas de définition du «temps» qui ne soit pas autoréférentielle, c'est-à-dire où la notion de temps est absente.

1. Les dates et heures

Introduction

Les dates sont un concept en apparence simple – on les utilise tout le temps sans trop y réfléchir – mais dont la manipulation contient de nombreuses subtilités. Or, en informatique, tout doit être explicite et rigoureux: on ne peut pas compter sur l'ordinateur pour interpréter les suppositions que l'on ferait dans le programme ou demander s'il a mal compris, contrairement à ce que ferait un humain…

C'est pourquoi l'on va disséquer ici les trois types de dates et leurs notions associées; la question des calendriers et celle de précision des dates; et enfin les détails propres à la gestion des dates en informatique.

1.1. Les 3 (trois ?!) types de dates

Parce que oui, il y a trois types de «dates», et leur confusion est la source d'une quantité infinie de problèmes en informatique.

1.1.1. Définitions

i Définition

Un **instant**, ou **moment** est **la position d'un évènement dans le temps**. Il est absolu [1]footnote:1 et indépendant de l'utilisateur.

Ce que ne laisse pas apparaitre cette définition, c'est que *le langage naturel ne dispose d'aucun moyen de désigner directement un instant*, mis à part des éléments trop relatifs de type «maintenant» ou paraphrases lourdes et spécifiques comme «à l'instant où s'ouvrit la porte». À la place, on utilise un concept différent, la date:

i Définition

Une **date** est **une indication de temps**, donnée dans un **calendrier** particulier avec une certaine **précision**. Elle peut être dépendante de l'utilisateur.

1. [2]footnote:1 On va laisser de côté ici le cas des relativités restreintes et générales qui casseraient ce paradigme. De toute façon, si vous devez vraiment les gérer, vous ne pourrez pas employer les bibliothèques existantes et standards des langages de programmation.

Ce flou dans la définition d'une date ainsi que des contraintes physiques vont entrainer la séparation du concept en deux entités distinctes.

Concernant la précision, on peut en première approche déterminer si une date est accompagnée d'une **indication horaire** ou non, mais en réalité l'éventail des précisions possibles est bien plus large. Les notions d'horaires font l'objet d'une section à part.

1.1.2. Les trois types de dates

1.1.2.1. L'instant

L'instant est à employer dès que l'on doit manipuler la position d'un évènement dans le temps.

C'est sans doute le cas le plus fréquent en informatique, et ne pose *théoriquement* aucun problème: si le langage naturel ne dispose pas d'outils simples pour déterminer un instant donné, l'informatique le permet.

Quelques exemples d'instants utilisés «directement» en langage courant: *«maintenant»* (très utilisé, mais très inutile sorti du contexte précis de son usage), *«Quand l'arbre fut foudroyé»* (désigne un instant très bien identifié… si on sait de quel arbre il s'agit).

1.1.2.2. La date indépendante de l'utilisateur

Dès que l'on doit présenter une information temporelle à un utilisateur humain, on doit employer une date. Le plus simple – pour le développeur – est de manier une de **date indépendante du référentiel de l'utilisateur**[3]footnote:2, donc une date **avec un fuseau horaire explicite** – indiqué par un décalage horaire.

Ceci évite d'avoir à connaitre ou supposer le fuseau horaire de votre usager et prévient les malentendus associés.

«Jeudi 17 juin 2021 à 20 h 00 UTC+2» est un exemple de date indépendante de l'utilisateur.

À propos des fuseaux horaires

Un fuseau horaire peut être défini par un *décalage horaire* (exemple: UTC+3:00) ou par une zone géographique, qui indique quelle heure légale s'applique (exemple: Europe/Paris). À cause des heures d'été et des modifications ponctuelles des heures légales, on ne peut pas coupler de manière stable un décalage horaire avec une indication géographique de fuseau horaire.
Dans la suite de ce contenu, pour alléger le texte et sauf indication contraire, un «fuseau horaire» s'entend **uniquement** comme identifié par un **décalage horaire numérique**.

2. [4]footnote:2 L'usage d'une date impliquant de recourir à un calendrier et un formatage particulier, toute date est obligatoirement tributaire d'un contexte local qu'il va falloir prendre en compte si votre logiciel a une vocation internationale. Mais une fois que l'utilisateur aura «décodé» le système de date que vous exploitez, la présence du fuseau horaire fera que la date ne sera pas ambigüe. Enfin, pas trop. Le moins possible, en tous cas.

FIGURE 1.1. – La carte des fuseaux horaires «normaux» (heures d'hiver le cas échéant) en application au 5 mai 2021. Domaine public .

1.1.2.3. La date locale, dépendante de l'utilisateur

Seulement, voilà, les dates indépendantes de l'utilisateur ont deux inconvénients:

1. elles sont très peu naturelles à manipuler;
2. certains évènements sont liés *à une date en tant que telle* et pas *à un instant* – pensez à une date d'anniversaire par exemple.

C'est pourquoi dans la plupart des cas, les saisies et affichages par l'usager se font via des **dates locales, dépendantes d'un référentiel utilisateur**. Ce référentiel peut être implicite (une application métier dont on sait qu'elle n'est employée que dans un fuseau horaire connu), ou paramétrable dans les préférences de l'utilisateur – une subtilité que votre programme devra gérer.

«Le 1er janvier 2000 à 0h00», bien qu'étant une date très importante symboliquement, est totalement tributaire de l'utilisateur et n'équivaut à aucun moment spécifique[5]footnote:3.

1.1.3. Conversion entre ces concepts

Jusqu'ici, tout peut paraitre simple. Naïvement, on pourrait se dire qu'il suffit de tout stocker dans l'un de ces formats, le plus pratique d'un point de vue technique, et de tout convertir dans le bon «type de date» quand on en a besoin. Seulement, la dure réalité est:

3. [6]footnote:3 Plus exactement, *«Le 1er janvier 2000 à 0h00»* correspond à autant d'instants qu'il y avait de fuseaux horaires «en activité» à ce moment là, chaque fuseau horaire ayant atteint *«Le 1er janvier 2000 à 0h00»* à un instant différent.

Il n'existe aucune conversion simple entre ces différents types de dates.
La confusion entre ces concepts, et les difficultés et erreurs de transformation qui en découlent, sont la source d'énormément de problèmes informatiques.

1.1.3.1. Conversion entre « instant » et « date indépendante de l'utilisateur »

Une date indépendante de l'utilisateur, si elle est complète, correspond toujours à un seul instant.

Par contre un instant peut être exprimé sous la forme de *plusieurs* dates avec fuseau horaire – une par fuseau horaire.

L'instant symbolisé par«le 4 mai 2005 à 03 heures 02 minutes et 01 seconde à UTC+0» est aussi représenté par:

— «le 4 mai 2005 à **04** heures 02 minutes et 01 seconde à **UTC+1**»;
— par «le **3** mai 2005 à **23** heures 02 minutes et 01 seconde à **UTC-4**»;
— ou même par «le 4 mai 2005 à **08** heures **47** minutes et 01 seconde à **UTC+5:45** » – parce que le décalage n'est pas obligatoirement un nombre *entier* d'heures.

1.1.3.2. Conversion qui implique une date dépendante de l'utilisateur

Une conversion entre d'une part, un instant ou une date avec fuseau horaire, et d'autre part une date dépendante de l'usager, se heurte aux problématiques de définition du fuseau horaire à utiliser.

Pour commencer, cette conversion **n'est pas possible** si la date ne vise pas à représenter un instant. Si mon anniversaire est le 4 mai, il est le 4 mai *quel que soit le fuseau horaire dans lequel je me trouve et quel que soit le fuseau horaire dans lequel se trouve mon interlocuteur.*

D'autre part, la récupération du décalage horaire à employer peut être particulièrement complexe:

— Il peut changer selon l'utilisateur de l'application.
— Il peut changer selon la date: l'utilisateur ne fournit pas directement un décalage horaire en «nombre d'heures et de minutes de décalage à appliquer», mais en «fuseau horaire légal à tel endroit». Or, certains pays ou régions utilisent des heures différentes l'été et l'hiver – ce qui doit être pris en compte.
— Pire, les dates d'application des heures d'été et d'hiver peuvent évoluer (et même apparaitre et disparaitre).
— De même, l'heure légale applicable dans un territoire donné peut changer au cours du temps. Dans les exemples très récents, la Corée du Nord est passée de UTC+9:00 à UTC+8:30 le 15 aout 2015, puis est revenue à UTC+9:00 le 5 mai 2018 – la Corée du Sud avait fait les mêmes modifications en 1954 et 1961.
— Enfin, si vous devez gérer des dates anciennes, la notion même de «fuseau horaire» est relativement moderne: elle n'existe pas réellement avant 1858, et n'est généralisée au monde entier qu'à partir de 1929.

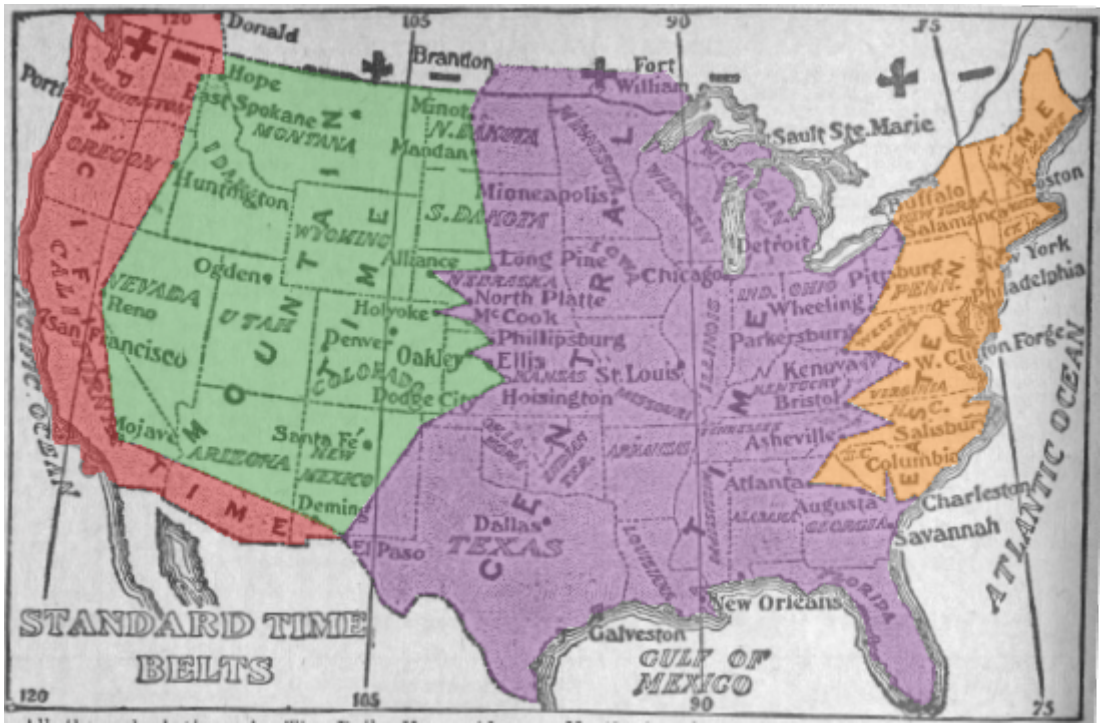

FIGURE 1.2. – Carte des fuseaux horaires en vigueur aux États-Unis d'Amérique en 1913, qui sont différents de ceux en application en 2021. CC–0 Owen Blacker

1.1.3.3. Un outil pour retrouver les décalages horaires : *tz database*

Lorsque l'utilisateur doit renseigner son fuseau horaire, ou quand on doit convertir des dates dans lesquelles interviennent un fuseau horaire se pose la question suivante: *quels sont les décalages horaires correspondants?*

On l'a vu plus haut: non seulement la liste à un instant T est longue et fastidieuse à maintenir, mais en plus elle change régulièrement au cours du temps!

Heureusement il existe un standard de fait, c'est **tz database**, aussi connue sous les désignations de **tzdata**, de **zoneinfo database** ou encore de **IANA time zone database**. Comme son nom l'indique, c'est une base de données qui identifie quels décalages appliquer en fonction de la zone géographique et de la date. Vous pouvez aller voir son site officiel ou la page Wikipédia anglophone (la version francophone est très légère).

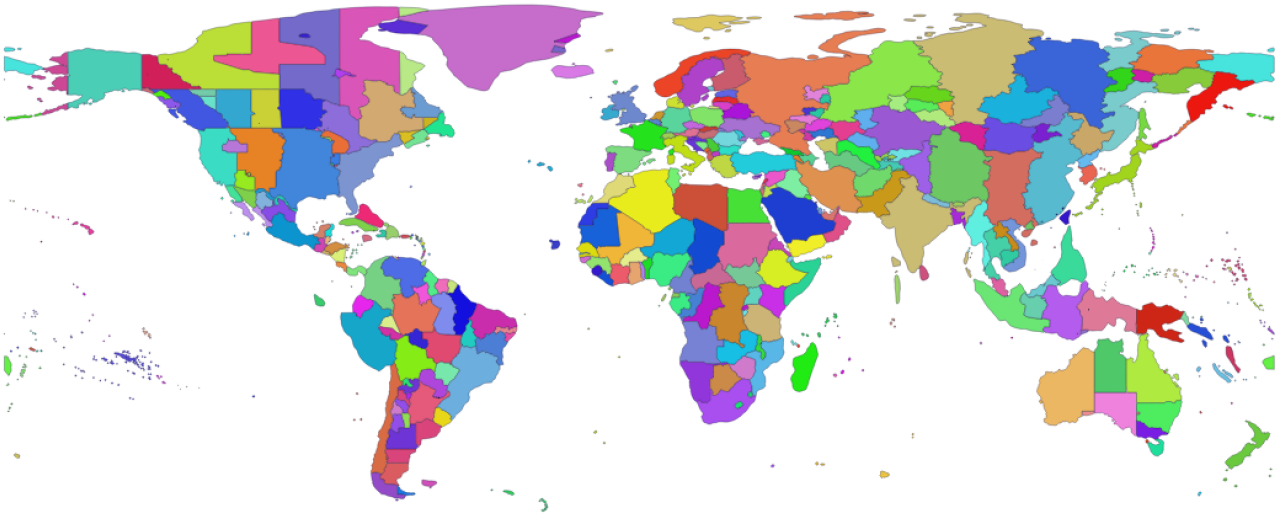

FIGURE 1.3. – La carte complète des zones tzdata, début 2017. Oui, il y a des zones sur l'océan, ça n'est pas une blague du cartographe – CC BY-SA 4.0 Evan Siroky

Cette base de données est très fiable pour toutes les dates entre 1970 et sa date de dernière mise à jour – l'IANA peut en publier plusieurs moutures par an. En dehors de ces bornes, elle peut contenir des erreurs, ce qui est une bonne raison pour la maintenir à jour. Heureusement, cette tâche est presque toujours confiée au système d'exploitation ou aux frameworks. Par exemple, les mises à jour Java embarquent les dernières versions de tzdata à parution . Cela dit, c'est toujours intéressant de vérifier *qui* est responsable de cette base de données dans votre système.

1.2. Digression sur les calendriers

Dans la grande majorité des cas, le seul calendrier à gérer sera le calendrier grégorien , qui est le standard *de fait* presque partout, et celui utilisé par défaut dans tous les langages de programmation.

Mais il se peut que vous ayez à gérer d'autres types de calendriers, pour des raisons culturelles, religieuses, ou tout simplement de date.

En ce qui concerne ce dernier point: si vous devez gérer des périodes historiques, le calendrier grégorien n'est réellement entré en usage que le 15 octobre 1582 dans certains pays, mais cette date varie beaucoup selon les régions du monde , et il n'est pas d'usage de l'appliquer de manière rétroactive. Cela donne des situations étranges, comme une nuit du 4 au 15 octobre 1582 en Espagne – le 4 octobre s'entendant selon le calendrier julien, et le 15 selon le calendrier grégorien.

Certains calendriers peuvent utiliser des logiques et des durées de mois et d'année très différentes du calendrier grégorien. En fait, certains calendriers peuvent être imprévisibles parce que basés sur l'observation d'un phénomène naturel.

Exemple: une date, de multiples façons de la nommer

FIGURE 1.4. – Une page de calendrier parue à Thessalonique en 1896. Domaine public .

Cette page de calendrier montre la date du jour pour le même jour dans six langues (turc ottoman, arménien, «hébreu-espagnol», grec, bulgare et français) et surtout cinq systèmes de calendriers différents. Ce jour-là était nommé:

— 20 Teşrin-i Evvel 1311 dans le calendrier rumi (calendrier julien de l'Empire Ottoman);
— 14 Jumādā al-Ūlā 1313 dans le calendrier islamique;
— 14 Cheshvan 5656 dans le calendrier hébraïque;
— 20 octobre 1896 dans le calendrier julien;
— 1er novembre 1896 dans le calendrier grégorien.

Toutes ces subtilités ont deux implications:

D'une part, à moins d'une très bonne raison – comme faire des calculs spécifiques à un calendrier – toute manipulation de dates supposant un calendrier devrait se faire sur le calendrier grégorien, avec une conversion dans un autre calendrier le plus tard possible, au plus proche de l'affichage à l'utilisateur.

D'autre part, employez des bibliothèques tierces pour gérer les conversions entre calendriers. Gérer *un seul calendrier* est déjà un champ de mines; prendre correctement en compte toutes les subtilités de transformation sans savoir très exactement ce que l'on fait, c'est la certitude de se tromper quelque part.

1.3. Précision d'une date et dates partielles

1.3.1. Une question de précision

Une date est toujours donnée avec une certaine **précision**, implicite ou explicite. Cette précision n'a rien à voir avec la *durée* éventuelle de l'évènement daté.

Par exemple, Iron Maiden :

— a été fondé en 1975 (précision à l'année, évènement instantané);
— a sorti son album «The Number Of The Beast» le 29 mars 1982 (précision au jour, évènement instantané);
— a commencé sa tournée «Somewhere Back in Time» le 1er février 2008 (précision au jour, évènement de plus d'un an);
— a joué un concert au Hellfest le 24 juin 2018 à 21 h 30 (précision à la minute, évènement de deux heures).

La principale difficulté est que cette précision est rarement *explicite*, et que la date indiquée peut être *plus précise* que la date réelle. Par exemple, cette réunion notée pour le 3 juin à 10 h 00 dans votre agenda ne commencera *probablement pas exactement* à 10 h 00 ce jour-là.

La précision peut aussi dépendre des connaissances de la personne qui mentionne la date: un admirateur saura que Iron Maiden a été fondé en 1975 (précision à l'année), un fan *hardcore* (ou qui vient de lire l'article Wikipédia) pourra préciser que le groupe a été créé le *25 décembre 1975* (précision à la journée), et un membre fondateur du groupe sera peut-être capable de préciser l'heure de cet évènement particulier. Mais personne ne pourra donner de précision *parfaite*, parce que par nature, un fait tel que «la création d'un groupe» n'est pas assimilable à un seul instant unique.

La conséquence, c'est qu'il faut choisir une représentation de date dont la précision est cohérente avec la date manipulée. Ça n'a aucun sens d'utiliser une précision à l'année pour traiter des dates de début de réunion; pas plus que ça n'a de logique de les gérer à la milliseconde près.

— Un format de date *pas assez* détaillé va mener à des soucis de précision et de cohérence assez évidents,
— Un format de date *trop* précis va surtout poser des problèmes d'ergonomie à l'affichage, et peut fausser les comparaisons.

Les outils informatiques ne comprennent en standard que certains niveaux de précision, en général:

— À l'année (la date est un simple entier),
— Au jour,
— À la seconde,
— À une subdivision de la seconde qui dépend du langage (généralement la milliseconde ou la nanoseconde, mais certains langages utilisent des subdivisions plus exotiques).

En cas de besoin d'une précision différente, le plus simple est d'employer une représentation de meilleure précision, et d'ignorer les subdivisions non pertinentes, en prenant garde aux effets de bord.

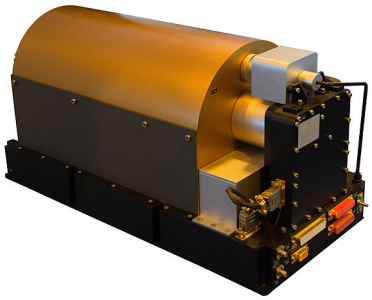

FIGURE 1.5. – Cette horloge atomique embarquée dans les satellites Galileo a une précision meilleure que 0,45 nanoseconde d'erreur sur 12 heures – ce qui est mieux que tout ce dont vous devriez avoir besoin un jour. CC BY 3.0 SkywalkerPL .

1.3.2. Les dates partielles

Du fait de leur précision, les dates sont toujours partielles, dans le sens où à partir d'un niveau de précision, les informations associées ne sont plus définies. Par exemple, sur une date précise à la journée, les heures, minutes, secondes et subdivisions ne sont pas déterminées.

Mais il peut arriver que **des éléments de date de niveau supérieur** soient ignorés, ce qui donne lieu à des choses qui *ressemblent* à des dates, mais qui n'en sont pas vraiment.

— *«Mon anniversaire est le 30 juin»* ressemble à une date avec seulement le mois et le jour du mois de définis, mais pas l'année.
— *«Mon réveil sonne à 7 h 30»* s'apparente à une date, mais avec uniquement l'heure et les minutes de définis.

Ces dates partielles sont utilisées pour les évènements récurrents, et n'ont généralement pas de représentation possible avec les outils standards des langages informatiques.

1.4. Usage des dates et heures en informatique

1.4.1. Les instants, la représentation sous forme de *timestamp*

1.4.1.1. Les *timestamp*, l'outil classique pour représenter un instant

Contrairement aux humains, l'informatique peut tout à fait représenter des instants précis sans l'intermédiaire d'une date. Soit directement via une classe (ou toute structure du langage) appelée `Instant`, soit plus souvent en passant par un ***timestamp*** [7]footnote:1

Un *timestamp* n'est rien d'autre qu'un nombre qui donne la quantité d'un intervalle de temps depuis un instant de référence bien connu baptisé *epoch* [9]footnote:2. En prenant l'instant de référence

1. [8]footnote:1 Une traduction française de «timestamp» serait «horodatage», mais je ne l'ai jamais vue utilisée dans le contexte de l'informatique.

et la durée écoulée depuis (qui peut être négative), on peut ainsi désigner n'importe quel instant sans avoir à faire appel à un système de date avec tous les problèmes que ça implique.

Et comme l'Univers est bien fait, tout le monde s'est mis d'accord pour employer le même intervalle de mesure des durées et le même instant de référence et… hahaha. Non. Évidemment que non.

Les différents systèmes de *timestamp*

Beaucoup de systèmes ont décidé de créer leur propre structure de *timestamp*, avec leurs instants de référence particuliers, unité de décompte du temps depuis cet instant, exigences associées (possibilité d'avoir des intervalles partiels ou négatifs)…
Donc, à moins d'avoir *la certitude* qu'ils utilisent exactement le même système, deux *timestamp* issus de deux logiciels différents ne seront pas compatibles entre eux.

Pourquoi un tel bazar? Parce que chaque technique a choisi son origine et son intervalle de temps standard en fonction de ses contraintes spécifiques: plage de dates qu'elle y a besoin de représenter, complexité des calculs à effectuer dessus, espace de stockage du *timestamp* à l'époque où l'informatique était beaucoup plus limitée qu'aujourd'hui…

Résultat, Wikipedia compte pas moins de 20 *epoch* différentes dont trois associées à des dates invalides (date inexistante comme le 0 janvier ou utilisation de l'an 1 du calendrier grégorien, qui a commencé en 1582).

Il y a quand même un format de *timestamp* qui est très largement employé, c'est l'heure Unix, ou heure POSIX, ou *unix timestamp*, etc . Sa valeur est le nombre de secondes depuis le 1er janvier 1970 à 00:00:00 UTC *hors secondes intercalaires*.

Exemple: récupérer l'heure Unix en ligne de commande sous Linux:

```
$ date +%s
1624144581
```

1.4.1.2. Intervalles de validité

Les *timestamp* peuvent avoir des intervalles de validité (premier et dernier instant représentables) assez limités. Ça n'est généralement pas un problème, *sauf si vous devez gérer des dates en dehors de la période contemporaine*. Les dates du futur peuvent aussi rapidement atteindre un seuil, comme le bug de l'an 2038 si vous utilisez le *timestamp* POSIX avec un entier signé de 32 bits (ce qui est la façon historique de le gérer).

Vous devriez donc toujours penser à vérifier les dates de validité des *timestamp* dès que vous l'employez pour manier autre chose que des dates proches de «maintenant».

2. [10]footnote:2 De l'anglais «époque» ou «ère», mais là encore la traduction française n'est pratiquement jamais utilisée dans le monde informatique.

1.4.1.3. Exemples de code

Java possède une classe qui représente un instant:

```
Instant maintenant = Instant.now();
```

On peut aussi récupérer directement le *timestamp* associé (qui ressemble au *timestamp* POSIX, mais n'est pas strictement identique , même si les raisons recevables d'utiliser cette valeur sont devenues rares:

```
// Ici en secondes depuis l'epoch Unix :
long timestamp = Instant.now().getEpochSecond();
// On trouve encore souvent l'ancienne méthode, qui donne des
    millisecondes depuis l'epoch Unix :
long timestamp = System.currentTimeMillis();
```

1.4.2. Les dates

Ici je vais être obligé d'être flou, parce que les solutions utilisées varient énormément selon le langage de programmation que vous employez (et parfois même *la bibliothèque* choisie au sein de ce langage).

1.4.2.1. Représentations internes et méthodes d'utilisation

Tout l'intérêt d'un objet (ou une structure, ou n'importe quoi d'autre selon votre langage) d'une date est de pouvoir **ignorer sa représentation interne**. Le but est de manipuler les dates **exclusivement à l'aide des méthodes dédiées**, qui vont gérer proprement tous les pièges et subtilités qui y sont associées.

Cette représentation interne et les méthodes rattachées sont complètement tributaires du langage de programmation. La seule constante, c'est que **la représentation standard des dates se fait dans le calendrier grégorien**. Si d'autres calendriers sont utilisables, c'est presque toujours une gestion de second ordre (via des conversions ou des appels explicites).

Par exemple, l'objet `LocalDate` de Java, qui décrit une date (sans heure ni fuseau horaire) dépendante de l'utilisateur emploie cette représentation interne:

```
    private final int year;
    private final short month;
    private final short day;
```

Mais son API n'expose pas directement cette représentation interne , en particulier on n'a pas besoin de savoir que `month` est de type `short` (aucune méthode de la classe ne renvoie de `short`),

et d'ailleurs la méthode `getMonth()` a un retour de type `Month`, qui est une énumération des mois existants du calendrier grégorien.

1.4.2.2. La représentation d'une date : la chaine de caractères

Ici, il me faut mettre quelque chose au point:

> **Chaine de caractères ≠ date**
>
> Une date exprimée sous la forme d'une chaine de caractères n'est qu'une **représentation** de cette date parmi d'autres. Les limites de cette représentation ne sont donc pas celles de la date elle-même.

En particulier, les problématiques d'internationalisation ou de tri ne se posent que lorsqu'on manipule **la représentation sous forme de chaine de caractères**, et pas la date elle-même – à l'exception du choix du calendrier pour certaines opérations.

La transformation d'une date en chaine de caractères ne pose généralement aucune question, à partir du moment où on a étudié ces deux éléments:

1. On connait le format de date exact voulu en sortie – en prenant compte de toutes les subtilités d'internationalisation susmentionnées;
2. On maitrise les méthodes de formatage de dates utilisées par le langage de programmation.

Pour ce second point, là encore: tout le monde a développé son propre système dans son coin, et on peut partir du principe que, en première approche, **chaque langage a son propre système de formatage incompatible avec les autres** – même s'il y a parfois des ressemblances.

Par exemple, si on veut déclarer un formatage de type «2021–06–19 12:34:56», Java demande d'écrire `yyyy-MM-dd HH:mm:ss` là où PHP demande d'écrire `Y-m-d H:i:s` et Go `2006-01-02 15:04:05` – trois systèmes complètement différents, donc.

La transformation d'une chaine de caractères en date (typiquement à la lecture de données externes au programme) exige de répondre aux mêmes questions, mais en plus de gérer correctement deux sortes d'erreurs:

1. Les formats invalides: quels sont les formats considérés comme valables à la saisie, et s'il y en a plusieurs, est-ce certain qu'une même entrée ne peut pas être comprise de deux façons différentes par deux formats acceptables?
2. Les chaines qui sont formatées comme des dates valides, mais qui ne correspondent pas à des dates existantes, par exemple un 0 janvier, ou un 29 février les années ordinaires.

Sur le second point, certains langages de programmation ont des méthodes de conversion qui sont *tolérantes par défaut*, c'est-à-dire qui vont corriger l'entrée pour la faire coïncider avec la date valide «la plus logique» (le 31 décembre de l'année précédente pour un 0 janvier, le 1er mars pour un 29 février d'une année non bissextile, voire le 30 juillet pour une saisie d'un 60 juin…). À vous de vérifier si ce comportement vous convient.

1.4.3. Opérations sur les dates

Il peut être tentant de faire soi-même les différentes opérations sur les dates, surtout quand on manipule un *timestamp* qui est un simple nombre. Sauf opérations triviales – il n'y en a pratiquement jamais lorsqu'on manie des dates –, c'est la garantie d'avoir une erreur ou un comportement étrange dans un cas particulier.

Ne faites pas d'arithmétique directement sur les dates

Ne faites jamais d'opérations directement sur les dates et *timestamp*. Utilisez systématiquement les méthodes dédiées de la bibliothèque que vous employez.
C'est beaucoup plus sûr, robuste, et le surcout est négligeable avec les machines actuelles.

1.4.4. Communication entre langages

1.4.4.1. La norme ISO 8601

Pour transmettre des dates entre des systèmes hétérogènes (c'est-à-dire quand on ne peut pas directement transmettre les types d'instants ou de dates), le moyen le plus simple et le plus fiable est d'utiliser la représentation adéquate **selon la norme ISO 8601** .

Cette représentation est:

— Normalisée, ce qui évite tout malentendu;
— Implémentée par défaut dans toute bibliothèque de traitement de date vaguement sérieuse – et d'ailleurs vous ne devriez jamais avoir à écrire le format ISO voulu à la main;
— Robuste, car elle gère les dates (avec les heures et toutes les subtilités) et les durées, y compris les récurrences.

Il suffit de se mettre d'accord sur la représentation ISO à employer entre les différentes parties, et hop! La communication se fait toute seule, sans surprise [11]footnote:3!

Si certaines représentations ISO sont assez lisibles pour des humains, ça n'est pas toujours le cas, et il vaut mieux éviter de trop exposer cette norme dans une interface utilisateur.

Quelques exemples de dates au format ISO 8601:

— Date seule: `2021-06-19` (date dépendante de l'utilisateur)
— Date et heure UTC: `2021-06-19T22:51:54+00:00`, `2021-06-19T22:51:54Z`, `20210619T225154` (date indépendante de l'utilisateur)
— Date et heure avec fuseau horaire: `1977-04-22T01:00:00-05:00` (date indépendante de l'utilisateur)

3. [12]footnote:3 Sauf quand l'une des implémentations les plus connues d'un langage a un bug dans son traitement de la norme ISO 8601 . Ici le moteur JavaScript V8, qui est utilisé dans Chrome et NodeJS, entre autres. Mais ça reste un cas rare.

Et un exemple de conversion d'une saisie au format ISO 8601 en une date puis en un instant en Java (`OffsetDateTime` représente une date avec décalage horaire explicite, donc indépendante de l'utilisateur). On voit que l'API standard suppose implicitement un format ISO 8601 par défaut en entrée.

```
1 OffsetDateTime date =
      OffsetDateTime.parse("2010-01-01T12:00:00+01:00");
2 Instant instant = date.toInstant();
```

1.4.4.2. API dédiées, conversions automatiques et SQL

Dans certains cas, la transmission entre systèmes informatiques se fait via des API dédiées qui n'utilisent pas de transformation explicite en un format textuel, et donc qui ne permettent pas l'usage de la norme susmentionnée. Un cas classique d'utilisation, c'est la communication avec une base de données, en particulier si l'API employée autorise le recours aux requêtes préparées et de leur passer directement des types temporels.

Ce genre de cas impose de vérifier que les types du langage et de la base de données sont bien équivalents, et que d'éventuelles transformations ne poseront pas de problème.

Un souci habituel, c'est la manipulation d'un instant représenté par une date indépendante de l'utilisateur. Il arrive que le langage appelant emploie une date au fuseau horaire local, par exemple *«le 19 juin 2021 à 01h00 UTC+02:00»*, mais que la base de données stocke cet instant au format UTC, ce qui donnera «le 18 juin 2021 à 23h00 UTC». Ce qui représente bien le même instant… mais cet instant sera associé au jour précédent si des calculs temporels sont faits directement par la base de données!

Conclusion

Derrière la notion floue de «date», se cachent donc trois réalités distinctes:

1. *l'instant*, qui désigne la position d'un évènement dans le temps, qui ne s'exprime presque jamais en langage naturel, mais qui possède des représentations informatiques;
2. *la date indépendante de l'utilisateur*, associée à un décalage horaire explicite;
3. *la date dépendante de l'utilisateur*, la plus proche de celle utilisée dans le langage naturel et qui peut *ne pas* équivaloir à un instant spécifique.

Les dates sont exprimées dans un *calendrier* particulier et ont une *précision* donnée.

En informatique, les instants peuvent être représentés directement ou par l'un des trop nombreux systèmes de *timestamp* existants.

Une date, en informatique, est une structure normalement distincte de sa représentation textuelle (sous forme de chaine de caractères). Cette représentation doit tenir compte des contraintes d'internationalisation. Transmettre des dates entre systèmes informatiques devrait se faire selon la norme ISO 8601.

Enfin, les opérations arithmétiques sur les dates ou *timestamp* devraient être proscrites en faveur des méthodes consacrées des bibliothèques utilisées.

2. Les durées

Introduction

Une durée, c'est un truc qui peut ressembler à une date, mais qui n'en est pas une, et qui pourrait être simple… si les systèmes d'unités utilisés pour les représenter n'étaient pas aussi délirants.

C'est ce qui va être détaillé dans cette partie, en terminant par les spécificités de l'usage des durées en informatique.

2.1. Une durée n'est pas une date (ça parait évident, mais ça va mieux en le disant)

Une date et une durée, ça se ressemble par bien des aspects. Une date comme une durée peuvent avoir des jours, des mois, des minutes, des secondes…

Mais c'est un piège! Les deux concepts sont fondamentalement différents. La date représente un point dans le temps; la durée représente la distance entre deux points dans le temps.

Pour symboliser un point (dans l'espace ou dans le temps), il n'y a que deux solutions possibles:

1. Énoncer une définition ad hoc – très pratique dans les conversations courantes, beaucoup moins dans un cadre formel donc en informatique;
2. Donner *la distance de ce point* par rapport à une référence bien connue.

La différence entre les mesures d'espace et de temps, c'est que pour l'espace, on utilise des choses comme la latitude et la longitude. Ces normes sont bien des distances par rapport à un référentiel (l'équateur et le méridien de Greenwich), mais les unités dans lesquelles on exprime ces intervalles ne sont pas les unités *habituelles* de mesure de distance de la vie courante.

Pour les dates, on donne là aussi la distance par rapport à un repère connu (la naissance théorique de Jésus pour les dates, et «minuit» pour les heures), *mais avec les unités de mesure de temps de la vie courante*. D'où une certaine confusion.

Et donc:

Une durée n'est pas une date

Si les concepts peuvent se ressembler, mélanger durées et dates va mener à des erreurs de calcul et des résultats qui n'ont pas de sens.
En particulier, **on ne peut pas utiliser une date pour exprimer une durée**.

2.2. Le grand bazar des unités de durée

Le plus grand problème de la gestion des durées, c'est que les unités utilisées sont très variées, sans lien cohérent entre elles, et pire: certaines ne correspondent même pas à une durée fixe.

2.2.1. Des unités très variées

Si le système décimal s'est imposé dans presque tous les domaines tant il est pratique, la mesure du temps a résisté à cette décimalisation.

Un calendrier décimal-mais-pas-trop a existé quelques années en France, mais comme l'année ne compte pas un nombre de jours multiple de 10, la décimalisation était partielle et le dispositif a fini par être abandonné.

Quant aux subdivisions décimales de la journée, elles ont été employées en Chine et sont toujours utilisées dans certaines applications précises [1]footnote:1, mais la norme est restée sur un découpage en vingt-quatre heures de soixante minutes .

On a donc, en partant de la seconde, unité de base du System International d'Unités :

— Des subdivisions décimales – dixièmes ou centièmes de secondes, millisecondes, microsecondes, nanosecondes, etc. La tierce n'est plus réellement usitée.
— Une minute qui vaut 60 secondes;
— Une heure qui vaut 60 minutes;
— Un jour qui vaut 24 heures;
— Une semaine qui vaut 7 jours.

Arrivés là on constate qu'on a plein de facteurs multiplicatifs différents selon le niveau de précision, et pire: que les multiples suivants n'utilisent pas des valeurs constantes!

1. [2]footnote:1 Principalement pour les calculs astronomiques, avec le jour julien – exploité aussi dans de vieilles bases de données, et qui n'a rien à voir avec le *calendrier* julien – et le système TLE de paramètres orbitaux à deux lignes . Microsoft Excel gère également les dates en tant que jours fractionnaires, mais normalement l'utilisateur n'a pas à le faire directement.

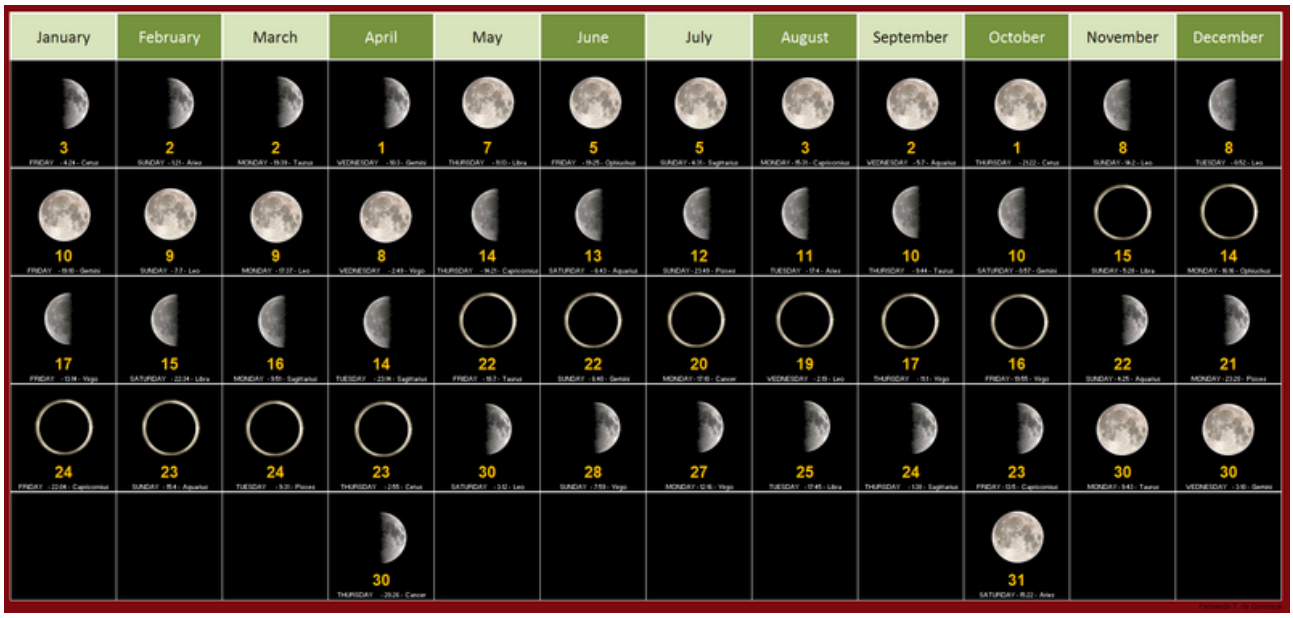

FIGURE 2.1. – Illustration: le calendrier des phases de la Lune en 2020. Notre satellite a longtemps servi, et sert encore, pour mesurer le temps.

2.2.2. Des unités qui ne représentent pas toujours la même chose

2.2.2.1. La longueur peut varier

On a un mois dont la valeur dépend… eh bien du mois que l'on considère, et donc du calendrier.

Avec un calendrier grégorien, un mois c'est: 28, 30 ou 31 jours, ou 29 mais ça n'est pas possible toutes les années. Avec d'autres calendriers, un mois peut prendre encore d'autres valeurs, et il peut exister des jours décomptés *en dehors* des mois.

Selon un calendrier grégorien, un an dure toujours 12 mois… mais 365 ou 366 jours, pour des raisons astronomiques. Mais cette durée peut être différente (pour *d'autres* raisons astronomiques !) avec d'autres calendriers.

Dans tous les cas, ni un mois ni une année n'ont un nombre entier ou constant de semaines.

Pour les durées plus longues qu'une année, on retombe souvent sur un système décimal, avec des décennies, siècles et millénaires (10, 100 et 1000 ans).

Ah, et dans certains cas on doit aussi prendre en compte l'imprévisible seconde intercalaire , qui vient casser la régularité de *tous les multiples de la seconde*.

2.2.2.2. La date de début peut varier aussi

Ceci concerne surtout l'année. L'année civile a une durée et un point de départ qui dépendent du calendrier utilisé. Mais il existe aussi *d'autres types* d'années, comme: l'année scolaire (qui ne dure pas une année civile, si on ne compte pas les grandes vacances), l'année fiscale dont les dates peuvent être spécifiques à une entreprise et qui peut durer jusqu'à 15 mois dans certains cas, l'année liturgique (ici dans la religion catholique, dont les dates de début et de fin – et donc la durée – sont variables), etc.

Manipuler une durée exprimée dans une unité plus grande qu'une semaine implique de savoir comment gérer les incohérences qui arrivent avec ce type d'unité!

2.2.3. Et ça n'est pas fini !

Il y a des unités qui sont encore plus floues, ce sont les unités associées à un contexte spécifique. L'exemple classique, c'est la conduite de projet, pour laquelle «une journée» ne vaudra pas 24 heures.

Non, parce que dans ce contexte, «une journée», c'est sans doute «le temps travaillé par un employé pendant une journée» (donc peut-être 7 heures, si vous êtes en France *et* aux 35 heures réelles, et autre chose dans tous les autres cas), et «une semaine», probablement «le temps travaillé par un salarié pendant une semaine»… ce qui est également très variable selon votre pays et votre entreprise. Ça peut même ne pas être un multiple entier de «une journée de travail» si l'organisation ne prévoit pas des journées identiques toute la semaine. Tout ça est peut-être paramétrable dans le logiciel, etc.

Il y a aussi toutes les situations où la législation s'en mêle, ainsi que les raisons historiques, et c'est l'occasion de vous montrer un exemple pratique.

Imaginons que vous ayez un système de paie à gérer, et que les employés sont rémunérés tous les mois. Un salarié a une ligne de paie qui ne doit pas être appliquée sur tout le mois, mais seulement sur une partie de celui-ci (disons qu'il est arrivé en cours du mois). La question est: *comment calculer la part à laquelle il a le droit*? Eh bien, il existe au moins **huit** façons de calculer «une fraction de mois», toutes étant employables en France[3]footnote:2! Ces huit façons de découper un mois sont détaillées sur ce site, avec des exemples chiffrés . Je vous laisse imaginer les prises de tête quand plusieurs de ces systèmes sont utilisés en parallèle.

2.3. Usage des durées en informatique

2.3.1. Des durées ? Quelles durées ?

Les durées sont souvent le parent pauvre des bibliothèques temporelles, quel que soit le langage de programmation choisi. C'est sans doute pour ça qu'on voit couramment des constantes en dur dans le code pour les représenter et les manipuler, comme:

```
1 public static final int SECONDS_PER_HOUR = 3_600;
```

2. [4]footnote:2 Pour l'anecdote, en France, dans beaucoup d'entreprises, ça n'est pas le même système de décompte qui est utilisé pour 1. les heures travaillées, 2. les congés payés pris normalement et 3. les congés payés pris en avance par rapport à leur droit théorique d'obtention. Souvent, poser des congés payés en avance compense exactement les heures travaillées – c'est le même système de calcul –, mais prendre des congés une fois leur droit acquis utilise un autre système de décompte légèrement plus avantageux!

Ce genre de code est généralement signe d'une arithmétique sur les dates et durées faite «à la main»… ce qui devrait vous alerter sur de probables bugs dans le programme qui les utilise.

2.3.2. Soyez surs de ce que vous faites en manipulant des durées

On l'a vu un peu plus haut, l'une des particularités des durées, c'est d'avoir des «unités» dont la taille réelle varie selon le contexte. C'est donc délicat à gérer et susceptible d'entrainer des bugs.

Avant de faire de l'arithmétique sur les durées et les dates, il est très important de bien se renseigner selon deux axes:

2.3.2.1. 1. Vérifier le besoin fonctionnel

Tout d'abord, **qu'est-ce qu'on demande exactement?**

Dès que l'on a besoin d'un calcul qui fait intervenir une unité de taille floue, quel est le comportement attendu?

Si on me demande d'ajouter un mois, est-ce que je dois ajouter un nombre fixe de jours? Est-ce que je dois tomber sur le même jour dans le mois suivant? Et que se passe-t-il si cette dernière définition n'est pas possible, comme «ajouter un mois au 31 mars»?

Avez-vous bien pensé à tous les cas tordus?

2.3.2.2. 2. Vérifier les hypothèses de votre framework

Ensuite, vérifiez les hypothèses de votre framework, et surtout qu'elles sont cohérentes avec votre besoin fonctionnel – surtout sur les cas aux limites.

D'une manière générale, évitez d'utiliser les unités mal définies tant que ça n'est pas strictement indispensable et que vous n'êtes pas certains du comportement obtenu.

2.3.2.3. Ne mélangez pas les choux et les carottes – pardon, les dates et les durées

Je sais, je l'ai déjà dit, mais je le répète. Un code tel que celui-ci doit vous alerter immédiatement:

```
1 public long projectRemainingTime(long currentDate, long
      remainingTime, long projectEnd) {
2     return currentDate + remainingTime - projectEnd;
3 }
```

Les problèmes qui devraient vous faire reprendre ce code sont:

1. `currentDate` n'est pas une date, mais un entier, donc probablement un *timestamp*, par conséquent est mal nommée (puisque ça n'est pas une date) voire pire (si c'est le mauvais type utilisé);
2. le calcul mélange allègrement des dates (les *timestamp* `currentDate` et `projectEnd`) et une durée. Dont on ne sait même pas s'ils sont exprimés dans la même unité!
3. on n'a aucune idée de ce qu'on renvoie (une date sous forme de *timestamp* ou une durée? En quelle unité ?) en lisant la méthode.

Une version plus propre de cette méthode serait quelque chose comme:

```
public Duration projectRemainingTime(LocalDateTime now, Duration
    remainingTime, LocalDateTime projectEnd) {
    return Duration.between(now, projectEnd).plus(remainingTime);
}
```

Ce qui est plus sûr, plus clair à la lecture (même si plus verbeux), et nous permet ne nous rendre compte de quelque chose: ce code a l'air de mesurer des différences de durées calendaires (jours de 24 heures), mais parle de projet. Est-ce qu'on ne voudrait pas des «jours de projets» dans ce contexte? D'ailleurs, combien d'heures dure «un jour de projet»? C'est des questions qui méritent d'être posées à qui de droit et documentées.

2.3.3. Communication entre langages – norme ISO 8601

Déjà évoquée ici , la norme ISO8601 permet de standardiser la représentation de durées. Seule cette norme devrait être utilisée pour communiquer des durées entre deux systèmes informatiques hétérogènes.

Par exemple, `P18Y9M4DT11H9M8S` représente une durée de 18 ans, 9 mois, 4 jours, 11 heures, 9 minutes et 8 secondes. Les années et mois utilisés sont calendaires.

Conclusion

Une durée est la distance entre deux points dans le temps – elle n'est donc pas une date et ne peut pas être représentée comme telle. Sa représentation passe par la combinaison d'un point de repère temporel et d'une «distance temporelle» à ce point.

La principale difficulté dans la gestion des dates est le système d'unités associé, qui conjugue:

— Des unités diverses avec des facteurs multiplicatifs différents entre eux;
— Des unités qui ont des durées imprécises et dont la valeur peut dépendre du contexte (calendrier, dates considérées, contraintes légales…).

Les représentations informatiques des durées sont souvent le parent pauvre des bibliothèques de manipulation d'informations temporelles, et il faut résister à la tentation de se lancer dans des calculs «à la main». Au contraire, tout calcul impliquant des durées mérite d'être vérifié pour être certain que ce qui est réalisé est bien ce qui était voulu – en plus d'employer les fonctions disponibles dans les bibliothèques. Et comme pour les dates, utilisez la norme ISO 8601 pour transférer des durées entre systèmes informatiques.

3. Les horloges

Introduction

Dans la vie courante, on utilise une horloge (au sens large du terme) dans deux cas distincts: soit pour connaitre une date, soit pour connaitre une durée, via la différence entre deux mesures sur l'horloge, si on n'a pas de chronomètre[1] footnote:1 pour ce faire. C'est pareil en informatique, mais pour des raisons techniques et historiques on a tendance à mélanger tout ça, ce qui peut provoquer des problèmes graves…

Aussi cette partie va essayer de rendre tout ça un peu plus clair. Puis elle abordera le point des planifications et celui de la gestion des horloges dans le cas particulier de l'informatique.

3.1. « Les » horloges ?

?

«Les» horloges? Comment ça, «les»? Il n'y a qu'une seule horloge, non?

Eh bien… non, et je ne parle pas de la technologie utilisée. En réalité, **deux types d'usages** nécessitent **deux types d'horloges différents**.

Le besoin habituel quand on traite d'horloge, c'est de connaitre *la date et l'heure actuelles*. Pour ça il faut, eh bien, une horloge qui donne la date et l'heure actuelles. Ça a l'air idiot dit comme ça, mais cette précision a des implications subtiles, mais qui peuvent conduire à des bugs gravissimes si elles sont mal comprises.

Une telle horloge, qui donne donc la date et l'heure qu'il est dans un lieu défini, est **l'horloge légale** – qui donne donc l'heure légale. Cette heure a la particularité *de ne pas être régulière dans le temps*. Je reviendrai sur les raisons de cette particularité et ses impacts dans la section dédiée.

Le second usage d'une horloge, c'est de mesurer un intervalle de temps. On a donc besoin d'une fonction de **chronomètre** qui est rarement nommée ainsi dans les outils informatiques. Pour mesurer une durée précise, on a besoin d'une horloge *régulière dans le temps*, et on vient de voir que l'horloge légale n'a pas cette propriété. Il nous faut alors un second type d'horloge, appelée **horloge monotone**, et que je vais détailler dans sa propre section.

1. [2] footnote:1 Théoriquement, l'appareil qui sert à mesurer un intervalle de temps entre deux évènements est un *chronographe* et pas un *chronomètre*. Néanmoins, comme c'est ce deuxième terme qui est utilisé dans le langage courant, c'est celui que j'emploierai ici.

3.2. L'horloge légale

Définition: l'horloge légale

L'**horloge légale** donne l'**heure légale**, donc celle qu'il est officiellement dans un lieu précis.
Elle renvoie **une date** associée à un fuseau horaire connu, mais parfois non explicite.
Elle est généralement couplée à **l'horloge du système**.

Ce type d'horloge s'utilise lorsque l'on a besoin d'accéder à une date et une heure légale, c'est-à-dire quand on a besoin de savoir quand est «maintenant».

Puisque l'horloge légale suit l'heure légale, elle hérite des bizarreries de ce genre de dispositif, comme l'heure d'été , les éventuelles secondes intercalaires ou toute autre implication de la loi sur l'heure légale. Une telle horloge renvoie donc bien une *date*, mais pas un *instant* précis.

Comme l'horloge légale est généralement couplée à l'horloge du système, elle en suit les variations. En particulier, si l'heure du système est réglée (manuellement ou à l'aide de NTP), l'heure affichée par l'horloge légale va inclure ces variations.

Ces deux caractéristiques font que rien ne garantit que l'horloge va avancer de manière régulière: toute intervention légale ou de réglage va provoquer des irrégularités. Par exemple, lors du passage de l'heure d'hiver à l'heure d'été, après 01:59:59, il est 03:00:00 – l'horloge saute d'une heure dans le futur. En fait, rien ne garantit que l'horloge légale *avance toujours dans le temps*. Tous les ans, au passage à l'heure d'hiver, elle passera de 02:59:59 à 02:00:00, et aura donc *reculé* d'une heure.

D'autre part, l'horloge légale va renvoyer l'heure légale *du système où est exécuté le programme* – ou parfois tout simplement l'heure UTC – ce qui peut être différent de l'heure utile pour l'usager. Le développeur doit penser à ça et prévoir les conversions idoines, surtout s'il n'a pas la main sur le réglage de cette horloge, comme dans le cas d'un programme exécuté sur un serveur ou dans le *cloud*.

Les documentations anglophones parlent souvent de *wall clock* à son sujet.

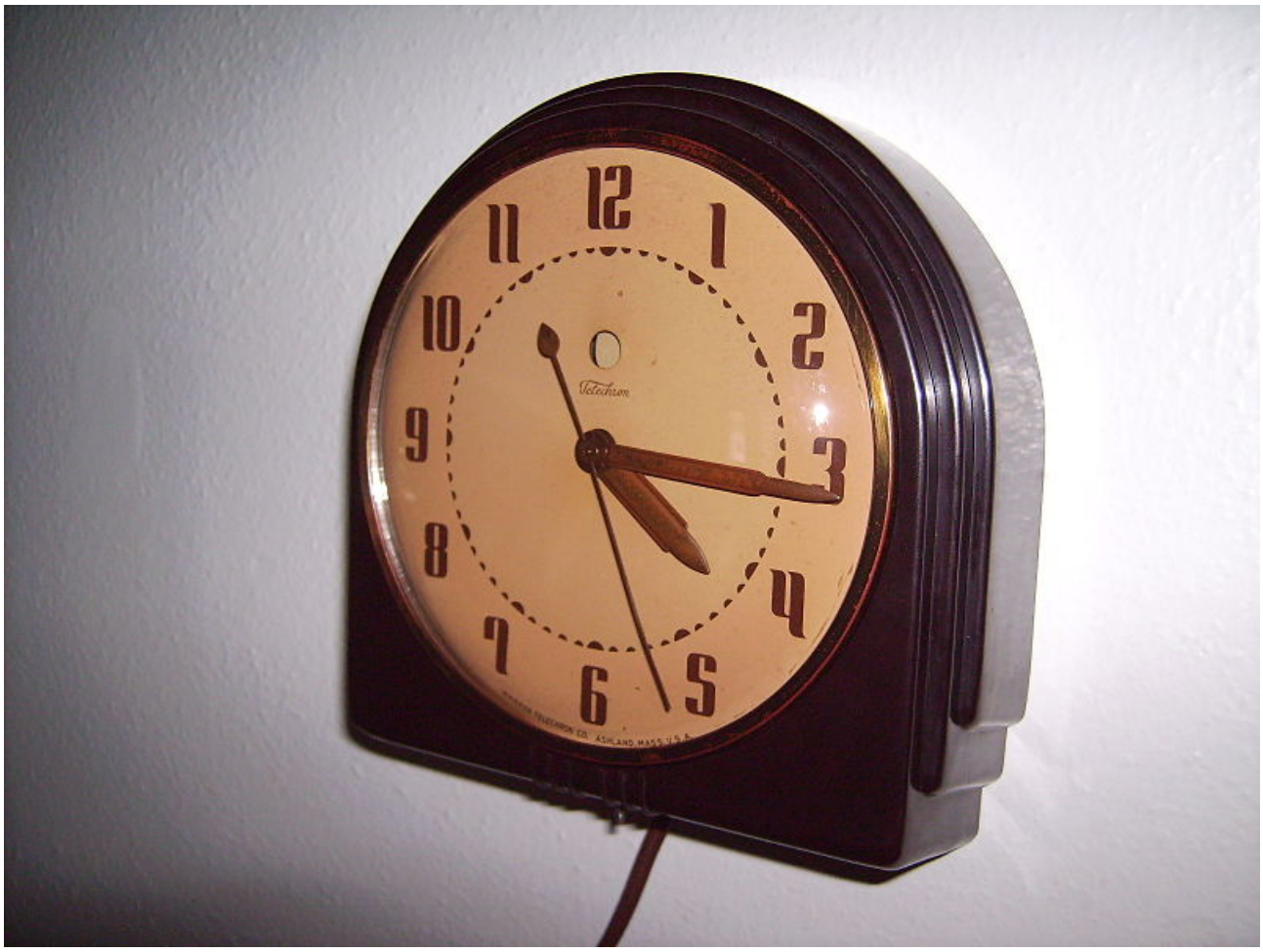

FIGURE 3.1. – Illustration: une horloge murale électrique. CC BY-SA 3.0 Wissenbourg .

3.3. L'horloge monotone

Définition: l'horloge monotone

L'horloge monotone n'est pas une horloge: c'est un **chronomètre** qui donne un **intervalle de temps écoulé depuis un instant de référence**.
Elle renvoie **une durée**.
Elle a **une avance régulière** et donc **indépendante de l'horloge système** et de tous les réglages ou fantaisies que cette dernière peut subir.

Ce type d'horloge sert pour toutes les tâches de chronométrage, comme mesurer le temps qu'a pris une action.

L'instant de référence de l'horloge monotone peut être «bien connu» (un repère fixe et universel) ou être totalement arbitraire (par exemple, l'instant de démarrage du programme). Dans le cas d'une horloge monotone dont l'instant de référence est arbitraire, il n'existe pas de moyen simple pour extraire des dates à partir des intervalles qu'elle fournit tant que l'on n'a pas déterminé d'abord la date à laquelle correspond l'instant de référence – ce qui peut ne pas être trivial.

La monotonie de l'horloge garantit que toutes les périodes mesurées seront les durées réelles et seront cohérentes entre elles, indépendamment des perturbations de l'horloge système. Cela évite les bizarreries telles que les durées négatives.

FIGURE 3.2. – Illustration: un chronomètre à main. CC BY-SA 3.0 Lesselich .

3.4. Le cas particulier des planifications

Une planification de tâche correctement utilisée nécessite de jongler avec les deux types d'horloges et les fuseaux horaires:

— L'horloge légale pour lancer les tâches à l'heure prévue;
— L'horloge monotone pour exécuter les tâches aux intervalles de temps voulus, **et** pour gérer les incohérences de l'horloge légale.

Admettons que je programme une sauvegarde tous les jours à 2h30 du matin.

— Si je ne me base que sur l'heure légale et mon fuseau horaire réel:
  — Lors du passage à l'heure d'hiver, elle va être lancée deux fois, parce que l'horloge légale remonte de 02:59:59 à 02:00:00
  — Lors du passage à l'heure d'été, elle ne va pas être lancée du tout, parce que l'horloge légale saute de 01:59:59 à 03:00:00
— Si je lance mes planifications à partir d'une horloge UTC, l'heure légale d'exécution va être décalée d'une heure pendant la moitié de l'année si l'utilisateur habite un lieu où s'applique une heure d'été (selon le moment où il a créé sa planification)[3]footnote:1.
— Si je lance mes programmes à partir de la seule horloge monotone, ça peut être difficile de savoir *quand* les lancer si celle-ci n'a pas de repère d'origine fixe.

Ajoutons à ça les difficultés habituelles des planificateurs, comme:

— la gestion des dépendances entre tâches;
— la gestion des collisions entre tâches (parce que l'occurrence précédente n'est pas finie, car trop longue ou démarrée à la main);
— la gestion des tâches qui auraient dû être exécutées, mais qui ne l'étaient pas parce que le planificateur n'était pas lancé.

Et on en arrive à cette conclusion:

! **Ne créez pas votre propre système de planification**. Trouvez-en un qui est connu pour fonctionner[5]footnote:2 et que vous pouvez utiliser directement, ça vous évitera de belles prises de tête.

1. [4]footnote:1 Un exemple de ce problème dans le monde réel: si vous avez un aspirateur robot de chez Ecovacs , ses tâches planifiées sont en réalité rattachées à des heures sur le fuseau horaire de Chine – c'est visible à un endroit dans l'interface – et donc se décaleront d'une heure aux changements d'horaires. Ainsi, la tâche que j'avais planifiée «juste après mon départ du boulot» s'est lancée «pendant le petit déjeuner»…

2. [6]footnote:2 Parce que certains planificateurs sont connus *pour ne pas gérer correctement* les problématiques susmentionnées, comme le planificateur Java IBM en 2010.

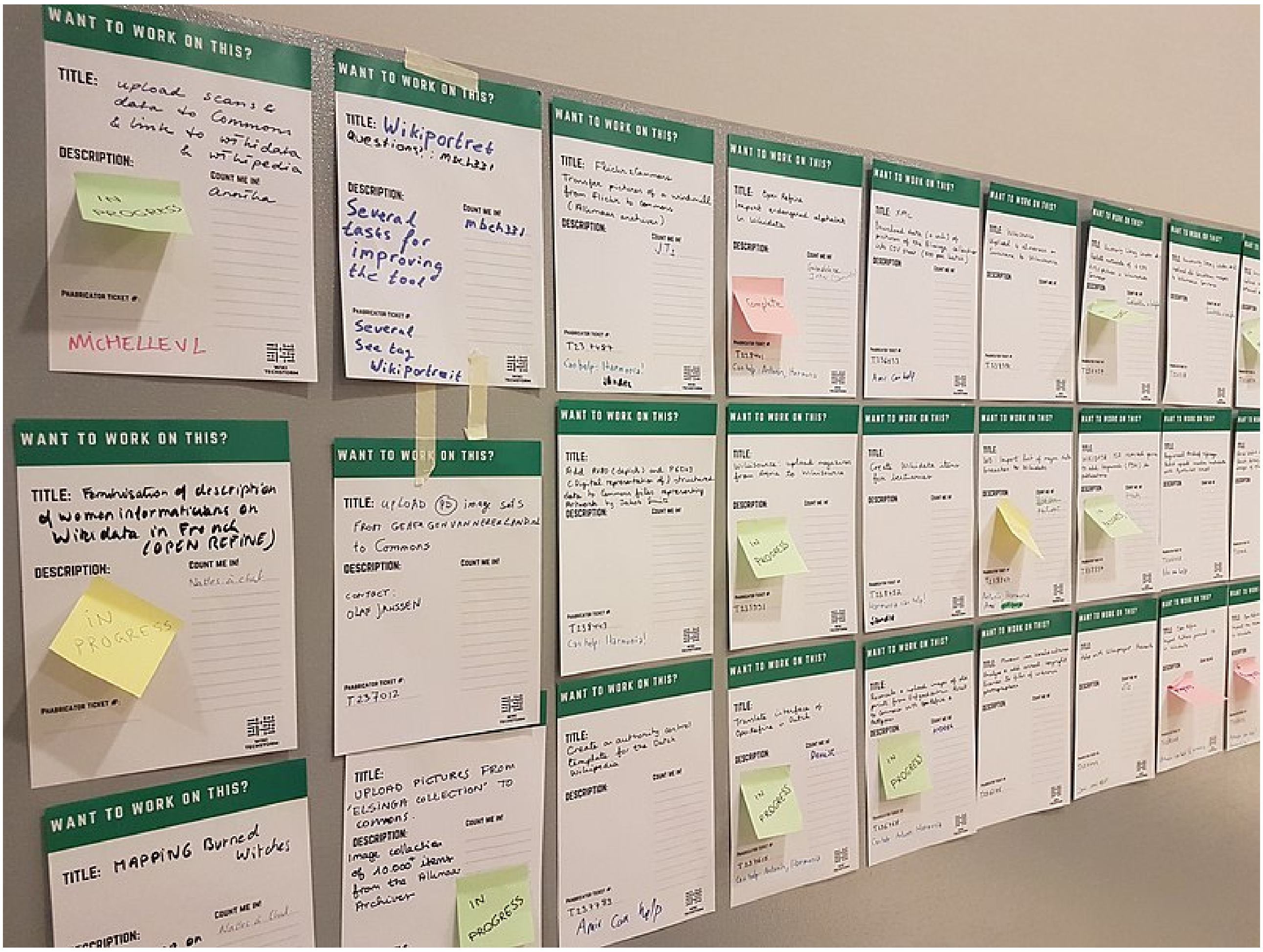

FIGURE 3.3. – Illustration: un tableau de planification de tâches, toujours plus efficace qu'un système informatique mal conçu. CCBY-SA 4.0 Ciell .

3.5. La gestion des horloges en informatique

3.5.1. Un peu de préparation

La première étape est d'identifier le type d'horloge dont vous avez réellement besoin.

La seconde étape, c'est de trouver **comment manipuler le type d'horloge voulu dans votre langage de programmation**, et là les choses peuvent se compliquer. La différence entre les deux types d'horloges est souvent mal comprise, et les concepts peuvent être mal nommés, voire complètement absents de certains langages ou de leurs vieilles versions.

En particulier, **les horloges monotones** sont parfois inexistantes ou d'accès peu intuitif.

i Comment trouver la documentation de l'horloge monotone dans votre langage

Essayer de rechercher le nom de votre langage plus:
— *monotonic clock*,

> — *high resolution time*,
> — ou *steady clock*
>
> Et dans tous les cas **vérifiez dans la documentation officielle** que ce que vous avez trouvé correspond bien à ce que vous cherchez!

Les annexes contiennent des liens vers la doc d'horloges monotones dans divers langages .

Vérifiez que vous utilisez les bonnes méthodes:beaucoup de langages ont «récemment» implémenté de nouvelles fonctionnalités pour gérer plus proprement et facilement toutes les subtilités susmentionnées. Une documentation ancienne pourrait vous laisser employer des méthodes obsolètes.

Pensez à vérifier *l'unité d'intervalle de temps* manipulée par votre horloge ainsi que *sa précision réelle*. Cette dernière peut être très différente de l'unité et varier selon le contexte. Par exemple, en JavaScript, l'horloge monotone renvoie des portions de millisecondes, mais la précision réelle peut être aussi mauvaise que 100 ms.

3.5.2. Les pièges à éviter

3.5.2.1. Le contexte utilisé par l'horloge légale

L'horloge légale peut renvoyer des valeurs différentes selon le contexte qu'on lui passe, en particulier le fuseau horaire désiré. D'autre part, si rien n'est précisé, la date renvoyée peut être dépendante du serveur. Par exemple en Java:

```
1  // Renvoie « maintenant », la date locale sur le fuseau horaire
       par défaut du serveur
2  LocalDateTime.now(); // En explicitant le fuseau horaire
3  DateTime.now();      // Avec fuseau horaire implicite
4  // Renvoie « maintenant », la date locale, sur le fuseau horaire
       UTC
5  LocalDateTime.now(Clock.systemUTC()); // En explicitant le fuseau
       horaire
6  DateTime.now(Clock.systemUTC());      // Avec fuseau horaire
       implicite
7  // « Maintenant » au Japon, de manière implicite
8  DateTime.now(ZoneId.of("JST"));
9  // Renvoie « maintenant », sous la forme d'un instant et pas d'une
       date
10 Instant.now();              // En tant qu'objet directement
       manipulable
11 System.currentTimeMillis(); // En tant que valeur de timestamp (à
       éviter)
```

Le système de dates et d'horloges de Java est particulièrement complet, votre langage ne permettra peut-être pas autant de subtilités.

3.5.2.2. La manipulation des horloges monotones

Le principal piège ici est que, dans l'immense majorité des cas, **ce que renvoie une horloge monotone n'est pas une date**. Il est donc **impossible** d'utiliser l'arithmétique et les conversions de dates immédiatement sur les retours de ces horloges.

Il est souvent indispensable de faire une soustraction entre deux appels à ce type d'horloge pour en déduire **une durée** que l'on pourra manipuler comme telle, les valeurs renvoyées par l'horloge n'ayant pas de sens pratique directement employable.

Par exemple en Java:

```java
long start = System.nanoTime();
var out = maMethodeLente();
System.out.println("maMethodeLente a été exécutée en " +
    ((System.nanoTime() - start) / 1_000_000.0 + " ms");
return out;
```

Conclusion

On a deux systèmes différents apparentés à une horloge à notre disposition:

— L'horloge légale, qui permet de savoir quand est «maintenant» et renvoie une date, sans garantie d'avance régulière.
— L'horloge monotone, qui est en réalité un chronomètre qui progresse régulièrement mais qui ne retourne généralement pas une date utilisable.

Les horloges en informatique sont souvent utilisées pour des tâches récurrentes ou de planification, ce qui est un exercice délicat: vous devriez toujours employer un planificateur tiers.

Enfin, manipuler les horloges en informatique est souvent assez simple, tant qu'on vérifie que l'on recourt aux méthodes adaptées au besoin et qu'on leur passe les bons paramètres.

4. Annexes

Introduction

Cette partie contient diverses annexes qui ne concernent pas *directement* le sujet principal, mais qui peuvent être utiles comme point de réflexion ou comme référence.

4.1. Afficher et faire saisir des dates : considérations sur l'expérience utilisateur

On l'a vu, les dates, c'est compliqué. Ce qui implique que les interactions avec les utilisateurs humains vont être complexes, ce qui mérite une annexe pour parler un peu de tout ça.

4.1.1. Afficher des dates à l'utilisateur

4.1.1.1. La taille compte, la précision aussi

Une date, ça peut être long, surtout quand on choisit un format d'affichage un peu verbeux:

> Dimanche 26 septembre 2021 à 23 heures et 18 minutes

Le choix de votre format d'affichage doit donc dépendre de la place disponible (et qu'il est raisonnable d'accorder) dans votre interface.

N'oubliez pas de prendre en compte les précisions des dates montrées: un simple *«2021»* sera de peu d'aide pour un rappel de rendez-vous, tandis qu'un *«dimanche 26 septembre 2021, 23:18:07.014»* sera un peu trop spécifique pour un rappel d'anniversaire.

Enfin, selon le contexte, ne montrer qu'une date partielle peut être une solution *à condition que l'on puisse toujours retrouver facilement la date complète en cas de doute*. Une page d'accueil de blog peut tout à fait indiquer que le dernier article a été publié *«le 3 juin»*, mais c'est important d'avoir l'information de l'année quelque part dans le texte, pour que l'utilisateur puisse vérifier que l'article n'est pas une vieillerie périmée.

4.1.1.2. La langue et le pays

Si votre application a une portée internationale, vous devrez gérer l'internationalisation des affichages de dates.

Le problème, c'est que les formats de date dépendent de la langue **et** du pays de l'utilisateur , et donc sont particulièrement casse-pieds à gérer.

Et ne pas les gérer – surtout si votre logiciel est disponible en plusieurs langues – c'est créer une confusion pour vos usagers. Aucun format de date n'est réellement universel et ne sera compris sans ambigüité partout. En particulier, le fameux format «anglais» mois/jour/année que l'on retrouve souvent dans les logiciels en anglais est en fait… un format purement nord-américain, et utilisé exclusivement aux USA et au Canada anglophone, et dans aucun autre pays anglophone.

Donc, si vous gérez un logiciel international, **laissez vos utilisateurs choisir leurs formats d'affichages de date** soit directement, soit en employant les standards de la langue et du pays qu'ils renseigneront.

C'est normalement assez simple à faire, parce que le système d'exploitation (sur ordinateur ou smartphone) fournit les formats à utiliser… sauf si vous développez pour du web – ce qui est quand même très courant.

4.1.1.3. Les dates relatives

Présenter des dates «relatives à maintenant» («Il y a X minutes», «Il y a X jours», etc.) peut être considéré comme pratique… ou très pénible selon les utilisateurs. C'est à vous de voir à quel point c'est intéressant dans votre application.

4.1.2. Faire saisir des dates à l'utilisateur

Si vos utilisateurs doivent saisir des dates, vous avez le choix entre utiliser un calendrier pour ce faire, ou les laisser taper les dates au clavier. L'idéal étant sans doute de proposer les deux fonctionnalités, avec l'accent plutôt mis sur l'une ou l'autre selon la destination de votre application. Une interface manipulée par le très grand public aura besoin d'un calendrier le plus clair possible; une interface professionnelle dans laquelle les usagers vont renseigner des dates au kilomètre va profiter d'une bonne zone de saisie texte.

4.1.2.1. Afficher un calendrier

Employez un vrai calendrier à l'ergonomie soignée, et pas une simple série de *dropdown* qui obligera l'utilisateur à des dizaines de clics et à fouiller dans une très longue liste pour trouver l'année qui l'intéresse.

Si votre système limite les dates valides à la saisie, indiquez clairement dans le calendrier quelles sont les dates valides et quelles sont les dates invalides.

4.1.2.2. Utiliser une zone de saisie de texte

Indiquez clairement le format attendu, quel qu'il soit! Si vous pouvez simplifier la vie des utilisateurs en lui évitant de saisir les séparateurs (exemple: l'usager entre `01022003` et l'interface affiche un joli `01 / 02 / 2003`, c'est encore mieux. Ça n'a l'air de rien, mais dans un progiciel ou toute autre application où on renseigne beaucoup de dates, c'est très appréciable.

4.2. Ressources par langages de progammation

4.2.1. C

4.2.1.1. Horloge monotone

— `clock_gettime` avec le paramètre `CLOCK_MONOTONIC` ou `CLOCK_MONOTONIC_COARSE` au moins sous Linux. Cette version n'est pas tout à fait monotone car affectée par les ajustements faits par `adjtime` ou NTP.

4.2.2. C++

4.2.2.1. Horloge monotone

`std::chrono::steady_clock` depuis C++11. L'origine du chronomètre dépend du contexte.

4.2.3. Go

4.2.3.1. Horloge monotone

`time.Now()` dans le package `time`. L'horloge peut ne pas être monotone en cas de mise en veille ou d'hibernation de la machine. L'origine du chronomètre est celle de l'heure POSIX (1er janvier 1970 à minuit UTC).

4.2.4. Java

4.2.4.1. Horloge monotone

`System.nanoTime()` depuis Java 1.5. L'origine du chronomètre est au démarrage de la JVM, et l'horloge peut ne pas être monotone en cas de mise en veille ou d'hibernation de la machine.

4.2.5. JavaScript / TypeScript

4.2.5.1. Horloge monotone

`performance.now()` qui renvoie un `DOMHighResTimeStamp` . L'origine du chronomètre dépend du contexte.

4.2.6. PHP

4.2.6.1. Horloge monotone

`hrtime()` depuis PHP7.3. L'origine du chronomètre est inconnue.

4.2.7. Python

4.2.7.1. Horloge monotone

`time.monotonic()` (depuis Python 3.3) et `time.monotonic_ns()` (depuis Python 3.7). L'origine des chronomètres est indéfinie.

4.2.8. Rust

4.2.8.1. Horloge monotone

`std::time::Instant` dont l'origine est inconnue.

Conclusion

Une fois ce tutoriel terminé, vous devriez maitriser les notions suivantes:

— Les trois types de «date» (instant, date indépendante de l'utilisateur et date dépendante de l'utilisateur), les concepts de calendrier et de précision des dates, et les subtilités de leur gestion en informatique: ce qu'est un *timestamp*, les outils de manipulation des dates (dont la différence avec une chaine de caractères et le formatage), l'existence de la norme ISO 8601.
— Ce qu'est une durée (et sa différence avec une date), les systèmes d'unités employés pour les mesurer et les écueils associés (en particulier les unités dont la taille est variable selon le contexte), et les difficultés de leur usage en informatique – notamment les pièges qui consistent à mélanger durées et dates, et à faire manuellement de l'arithmétique trop simpliste. Et encore une fois la norme ISO 8601.
— La distinction entre une horloge légale et une horloge monotone (qui est en fait un chronomètre), dans quel cas utiliser chacune d'elle et comment les exploiter en informatique – surtout dans le cas d'une planification.

Merci à @Renault, @QuentinC, @Aabu, @entwanne et @ToxicScorpius pour leurs retours en bêta-lecture, et à @entwanne pour la validation!
