ACE

Accelerated Capability Environment

Powered by

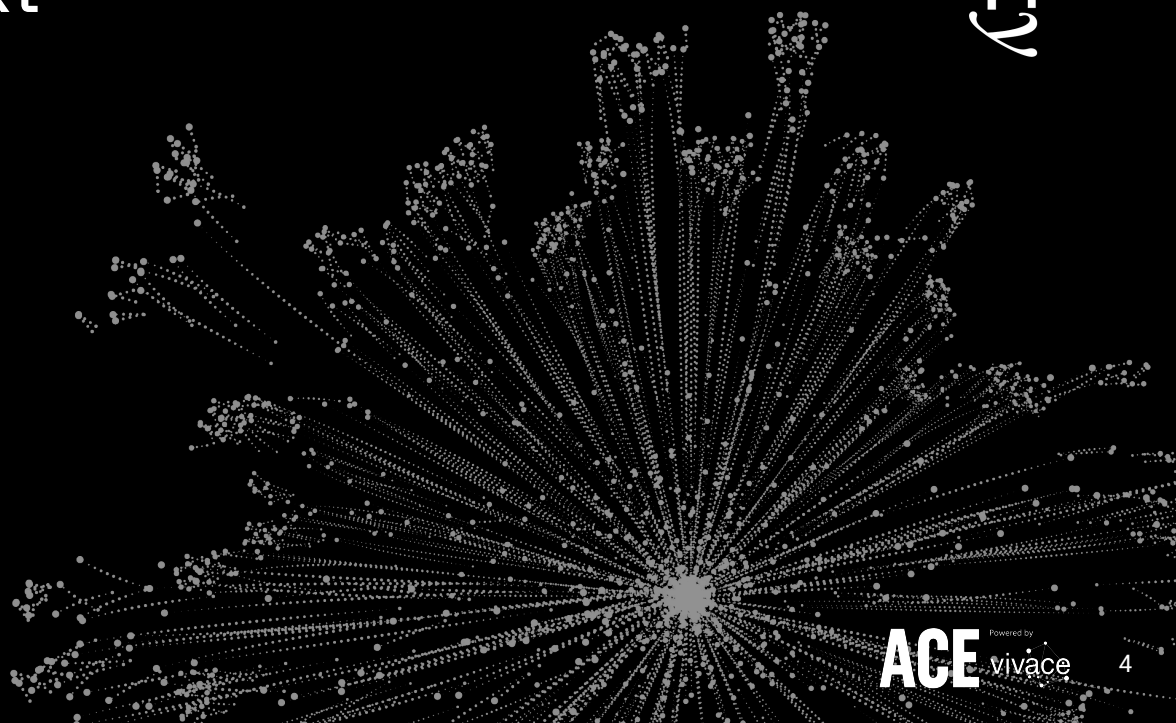
28/04/2021

C245 NHSX Agent Simulation Final Report

Version 2

Navigating this document

1. Project context



Definitions

Agent-based simulations (ABM): an agent based simulation attempts to create complex and emergent behaviour, through modelling a system as dynamically interacting rule-based agents. They consist of three layers:

- ➔ Agent layer - autonomous decision-making entity;
- ➔ Environment layer - the space in which the agents exist;
- ➔ Intelligence layer - the definition of rules that dictate the agents movement through the environment

Patient Pathways: a patient pathway is the specific route that a particular patient (*agent layer*) takes from the first referral request received date of a service request (*environment layer*), or the activity date of the first emergency activity where there is no related service request*.

**The intelligence layer in this instance governs the journey of the patient through the environment layer(s).*

Abstract

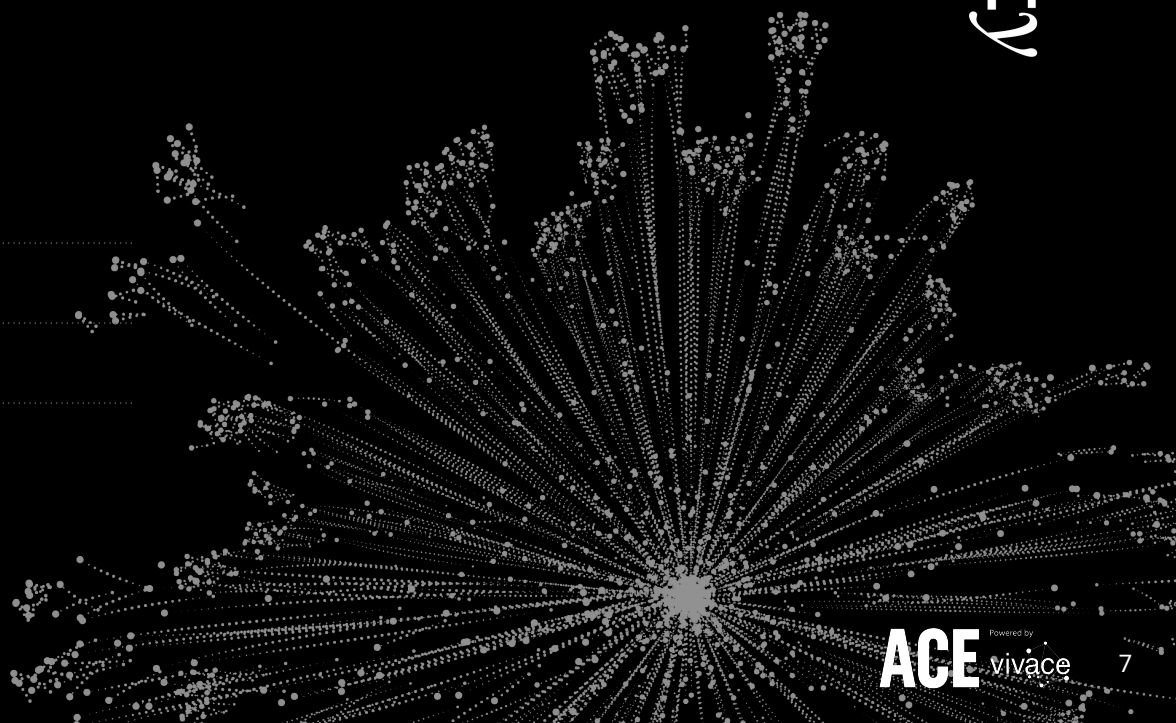
A patient pathway is the route taken by a patient through NHS services in order to address one or more health condition(s), and is reflected in a patient's health record. Due to sensitivities with data privacy and information governance, the NHS is not able to access sufficient data on patient pathways. Improved data access would be beneficial as there are numerous tools and services that support patient pathways, and so optimising their performance would provide advantages for both healthcare professionals and patients.

Since good quality, real patient pathway data is scarce, an alternative solution is to generate realistic synthetic data. In this project, Faculty were tasked with developing a framework for generating synthetic electronic health records that captured a patient pathway. The data would be produced by simulating a patient's interactions with various NHS services and, at each stage, updating their health record.

This report documents the groundwork for this project: a framework for a patient pathway agent-based model (ABM). Accompanying source code is located in the NHSX GitHub repository https://github.com/nhsx/SynPath. The code implements an "alpha data model" for the patient agent and its environment (e.g. a hospital), as well as functionality to run a simulation with logging. At the end the patient "record" is converted to a FHIR data format. There is a template language for the "intelligence layer" which governs the patient - environment interactions. This intelligence layer needs to be filled in by the user, depending on the pathway and dynamics they wish to model. Documentation and examples are provided in the repository.

The report also contains suggestions for improvements and future uses of this tool. Both the report and codebase were written by Faculty.

2. Overview of the data model



Key principles underpinning PoC Alpha data model and template language

1. Designed to support the creation of **longitudinal health care records**

2. The only **direct interaction** is between a Patient agent and its Environment

3. An interaction is an event which results in the **update of a Patient state** (this could occur even if a Patient is not "physically present" in an Environment, e.g. a multi-disciplinary team meeting)

4. The setup differs from a typical ABM in that **each simulation has a single central agent** - the Patient Agent. Therefore there is **no direct Patient-Patient interaction**

5. A Patient Agent only interacts with other Patients **indirectly**, namely via the parameters of the Environment (e.g. low hospital capacity may increase waiting times for patients). In the Alpha data model, we have implemented a **static version of the Environment** layer which has no temporal dynamics (adding these is a recommendation for future enhancements and detailed in section 6)

6. Typically agents in an ABM have **internal or private variables**. These have **not been incorporated for the Alpha data model** (but are a recommendation for the future)

7. We have only considered interactions (i.e. Patient state updates) involving exactly **one Patient and one Environment**. Future versions could consider multiple Environments in an update, or no Environments, or even no Patient.

Initialising the Patient Agent consists of inputting patient data either via the template language or manually from FHIR

Template language - main method

{json}

```
patients:
<patients_attributes>,

environments:
<environments_attributes>,

intelligence_dir:
</path/to/intelligence/dir>,

save_dir:
</path/to/save/dir>,

stopping_condition:
<stopping_condition>
```

Patient medical data - alternative method

HL7v2 *or* CSV

Conversion

Conversion not included in Alpha data model - future recommendation

HL7 FHIR V4

Selection of FHIR resources from:
Layer 1 - foundation
Layer 2 - base
Layer 3 - clinical

`.from_fhir`

PatientAgent

Required
```
id : str
gender : str
birth_date : datetime
```

Optional / internally set
```
start_date: datetime
conditions : DataFrame
medications : DataFrame
actions : DataFrame
record : list
```

New attributes can be added - either supply (key, value) in template script, or just key, and expect value to come .from_fhir

The below diagram illustrates how an end-to-end simulation of a patient pathway (for one patient) can be run using the data model and template language

1 *Load and parse config script*

Module: `patient_abm.simulation.template`
Function: `parse_config`

- Load config.json
- Parse 'patients' and 'environments' (i.e. if pointing to CSV or JSON files, load content from these files)
- Validate config

Module: `patient_abm.simulation.run`
Function: `simulate`

Intelligence layer

3 *Run patient pathway simulation for all patients in a loop*

`for patient in patients:`
Module: `patient_abm.simulation.run`
Function: `run_patient_simulation`

(See next slide for details of run_patient_simulation)

Outputs

Saved in the config `save_dir`:

```
<simulation_id_0> /
   agents /
       patient_<patient_id_0>.tar
       environment_<environment_0_id>.tar
       environment_<environment_1_id>.tar
       ...
   fhir /
       bundle.json
   main.log
   patient.log

<simulation_id_1> /
   agents /
       patient_<patient_id_1>.tar
       environment_<environment_0_id>.tar
       environment_<environment_1_id>.tar
       ...
   fhir /
       bundle.json
   main.log
   patient.log
```

`config.json`

```
patients:
<patients_attributes>,

environments:
<environments_attributes>,

intelligence_dir:
</path/to/intelligence/dir>,

save_dir:
</path/to/save/dir>,

stopping_condition:
<stopping_condition>
```

(Optional) Refer to patients and environments attributes data from JSON or CSV

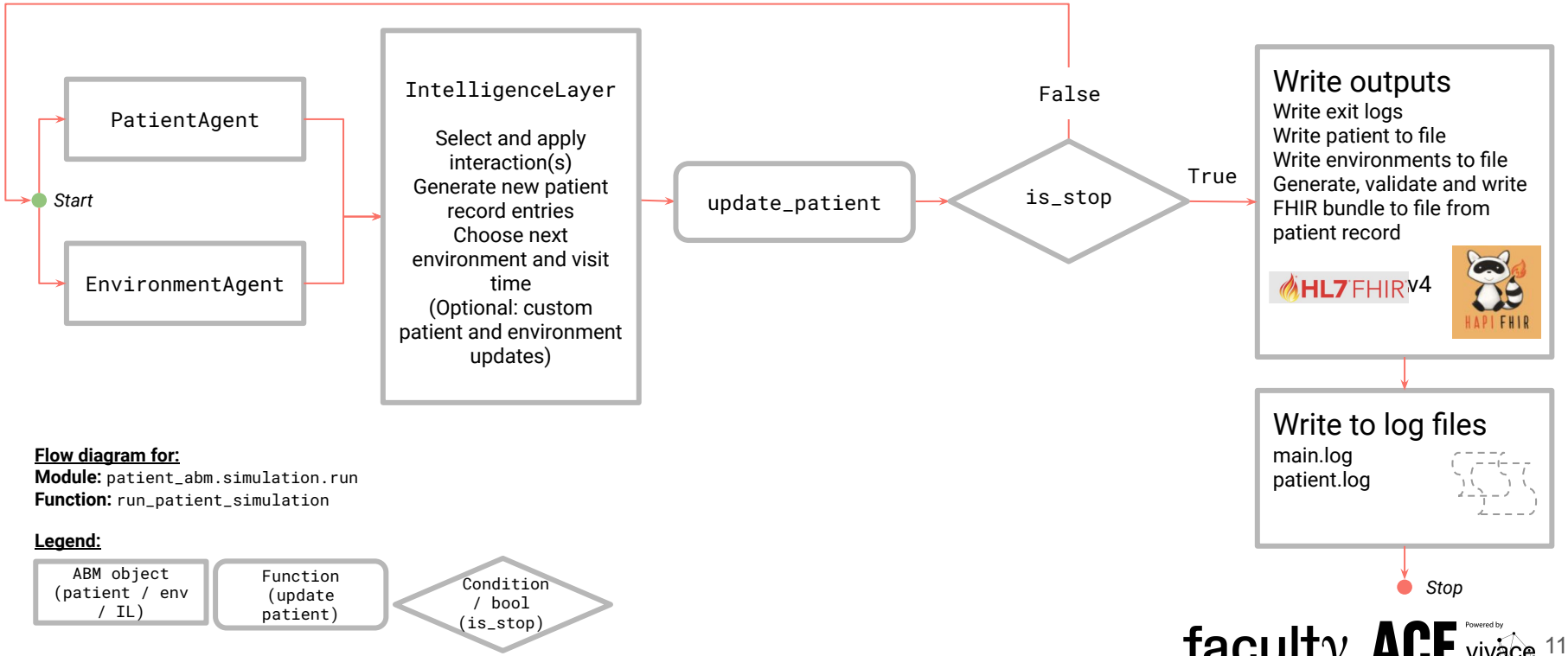
{json}
csv

2 *Instantiate objects from config.json*

Module: `patient_abm.simulation.initialize`
Function: `initialize`

- Create PatientAgent and EnvironmentAgent instances according to config parameters
- Load intelligence and interaction function handles
- Load all other simulation parameters

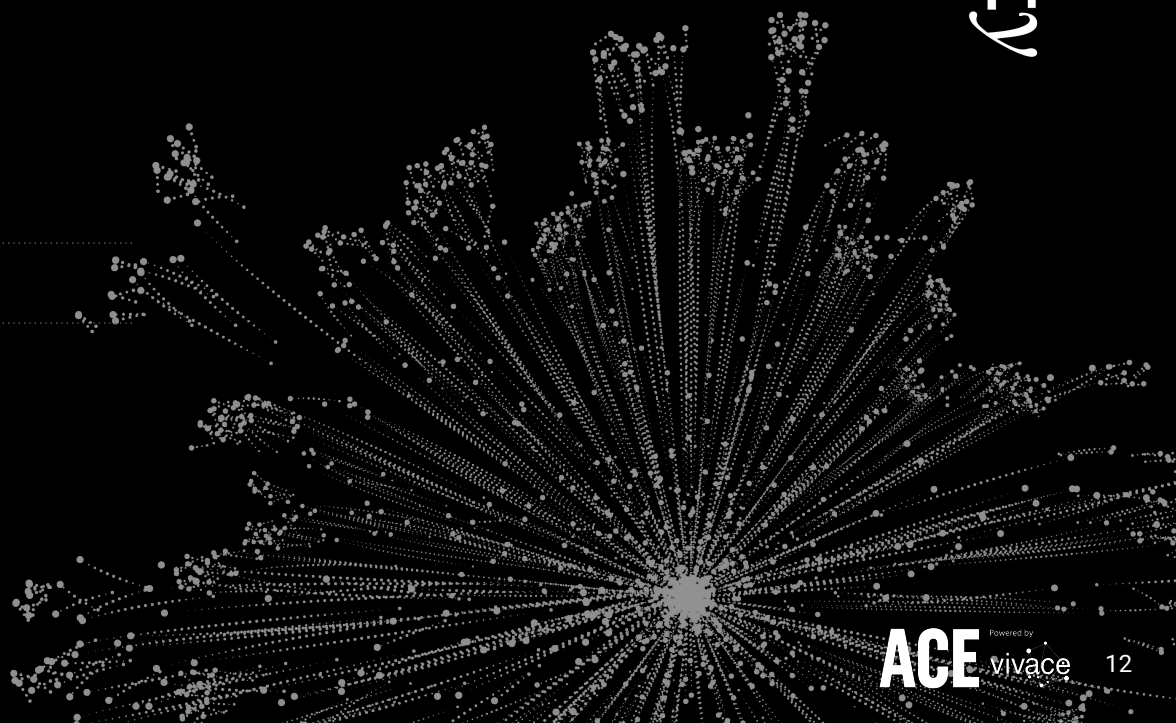
The below diagram provides further detail on how the Patient state is updated during the simulation



3. Patient Agent

Design features

Recommendations for the future



For the Alpha data model and template language, the Patient Agent incorporates the following attributes and methods:

PatientAgent

Attributes (Patient Agent properties)

`patient_id: Union[str, int]` Patient ID, string or integer type (required)
`gender: str` Patient gender, string type (required)
`birth_date: Union[str, datetime.datetime]` Patient birth date, can be datetime or string type (required)
`start_time: Union[str, datetime.datetime]` Start time of simulation in patient time
`name: Optional[str]` Patient name, by default None. Defaults to patient_id if not supplied
`conditions: Optional[Union[pandas.DataFrame, List[dict]]]` Patient conditions (problems or diseases)
`medications: Optional[Union[pandas.DataFrame, List[dict]]]` Patient medications
`actions: Optional[Union[pandas.DataFrame, List[dict]]]` Patient actions that they might have outstanding
`conditions_custom_fields: List[str]` Additional custom fields to add to conditions table, e.g. ["severity"]
`medications_custom_fields: List[str]` Additional custom fields to add to medications table, e.g.["side_effect_experienced"]
`actions_custom_fields: List[str]` Additional custom fields to add to actions table, e.g. ["delay"]
`inpatient: Optional[dict]` Information about the patient's inpatient status
`alive: bool` Whether the patient is alive
`record: List[dict]` The patient record. Can start non-empty if patient has existing record. If existing record has e.g. Condition entry, then conditions table is populated
`patient_profile: Optional[dict]` Patient demographic profile. Essentially maps to FHIR Patient resource and becomes the first entry in the patient record
`**kwargs` Keyword arguments, set as attributes. If key starts with prefix 'patient__' it will also get added to the patient_profile (with prefix removed)

Methods (Patient Agent functions)

`update` Update patient record with new entries, validate record for duplicates and time-ordering, and update conditions, medications, actions tables
`save` Save patient agent as tar file
`load` Load patient agent from a tar file

For the Alpha data model and template language, the Patient Agent incorporates the following design features

- **Key role of the Patient Agent**: an identifiable entity that stores a coherent electronic health record about itself.
- **Patient Agent properties**: to make the Patient identifiable, it is assigned a patient_id, gender and birth_date as required inputs. These attributes were selected because they could significantly impact the outcomes of their simulated pathway, and we were aiming for a minimum representation of the Patient Agent in the Alpha data model. There is flexibility for adding other physical properties to the Patient (e.g. weight) by supplying this information as keyword arguments in the Patient class.
- **Patient Agent health record**: the central attribute of the Patient Agent is its record, and other attributes are derived from the record, such as the patient's conditions, medications, or any outstanding actions. The record is a list of entries, which are FHIR-like resources, but converted to a simpler representation so that the user does not have to write raw FHIR in the Intelligence layer, but structured enough to produce a FHIR bundle form the record.
- **Patient Agent record updates**: when new entries are generated by the Intelligence layer, the simulation automatically adds them to the Patient via its update method. This appends to the existing record if certain validation checks are passed (e.g. duplication). Subsequently, the conditions, medications,and outstanding actions are updated.
- **Patient Agent new attributes:** there is flexible functionality for the user to add new features to the Patient Agent. New attributes are easily added via kwargs. For more complex attributes, such as adding a "severity" column to the conditions table, there is a parameter called conditions_custom_fields. This is a list where the user can add new fields like "severity" to the conditions table, and then entries containing information about "severity" will automatically populate the table.
- **Assigning required attributes:** some new attributes may be considered common features and therefore required, for example pregnancy status - this also presents the possibility that the patient may give birth during the simulated pathway, which would involve the creation of a new patient agent mid-simulation. This is a question to tackle when implementing multi-patient pathways (see future enhancement P4).

In future, we recommend the following enhancements to the Patient Agent:

	Enhancement	Complexity
P1	Expand list of FHIR resources / fields that the interaction layer can update in the Patient Agent in the FHIR record	Easy
P2	Incorporation of additional Patient Agent attributes via the PRSB Core Information Standard (mapped to FHIR)	Medium
P3	Capturing of hidden ('latent' / 'private') variables	Medium
P4	Expansion to multi-agent simulations, where multiple patient pathways can be simulated concurrently	Hard

The following slides provide recommendations on how the above enhancements could be implemented.

P1. Expand list of FHIR resources / fields that the interaction layer can update in the Patient Agent in the FHIR record

Complexity

Easy

- To save the user from writing new patient record entries in raw FHIR format in the Intelligence layer, we have created an internal representation of FHIR resources. These are dictionaries, which can be used when converting the patient record to FHIR.
- Currently conversion to / from the following FHIR resources is supported:
  - Patient
  - Encounter
  - Condition
  - Observation
  - Procedure
  - MedicationRequest
  - ServiceRequest
  - Appointment
- Within those, only certain fields are converted. **This is a simplified version, but more complexity could be added in order to produce richer FHIR outputs** (e.g. including more fields and data) and linking between resources (e.g. link to encounter resource from other related resources).

P2. Incorporation of additional Patient Agent attributes via the PRSB Core Information Standard (mapped to FHIR)

Complexity

Medium

- The PRSB Core Information Standard https://theprsb.org/standards/coreinformationstandard/ defines how to share health information in a structured manner. The site contains a link to an Excel spreadsheet which presents the types of information that can be collected about a patient - an extract is shown to the right.
- This kind of information can be **included in the Alpha data model, by mapping to an appropriate FHIR resource type.**
- Blue rows mark the start of a segment of information (Patient demographics, GP practice), it is possible to map these to FHIR resource types 'Patient; and 'Practitioner.' We believe it is also possible to map other core information segments to FHIR resource types.

Core information standard Version 1

Name	Conformance	Cardinality	Description
Person demographics	M	1 … 1	The person's details and contact information.
Person name	M	1 … 1	The full name of the person.
Person preferred name	R	0 … 1	The name by which a person wishes to be addressed.
Date of birth	M	1 … 1	The date of birth of the person.
Gender	R	0 … 1	The person's stated gender.
Person alias	R	0 … *	Other name(s) associated with the person.
Ethnicity	R	0 … 1	The ethnicity of the person as specified by the person.
Religion	R	0 … 1	The religious affiliation as specified by the person.
Sex	R	0 … 1	The person's phenotypic sex. Determines how the person will be
NHS number	R	0 … 1	The unique identifier for a person within the NHS in England and
Other identifier	R	0 … *	Country specific or local identifier, e.g. Community Health Index (
Person's address	M	1 … *	Person's usual place of residence, and where relevant temporary
Person's email address	O	0 … 1	Email address of the person
Person's telephone number	R	0 … *	Telephone contact details of the person. To include, e.g. mobile, v
Communication preferences	O	0 … 1	Preferred contact method, e.g. sign language, letter, phone, etc. Also preferred written communication format, e.g. large print, brai
Place of birth	O	0 … 1	The town and country of birth of the person.
Marital status	O	0 … 1	An indicator to identify the legal marital status of the person.
GP practice	M	1 … 1	Details of the person's GP practice
GP practice record entry	R	0 … *	This is an GP practice record entry. There may be 0 to many recc made up of a number of elements or data items
GP name	R	0 … 1	The name of the person's GP.
GP practice details	R	0 … 1	Name and address of the person's registered GP practice.

- Since the core information standard is mappable to FHIR, it can be used as an input to the Alpha data model. **We have documented in the code base how various FHIR resources can be supplied to the model when initialising the patients and environments**. This information can then be utilised by the Intelligence Layer.
- The core information standard may introduce some extension fields to the respective FHIR resource, as well as some additional constraints (e.g. Patient demographics must have a name field). For these reasons, the FHIR patient health record that is generated by the model may not be Core Information Standard compliant (even when it is FHIR compliant).
- In order to make it compliant, it would be possible to build the relevant rules into the FHIR report generator function; we also recommend building a validator for outputs.

P3. Capturing of hidden variables

Complexity

Medium

- Hidden variables are typically used in agent-based simulations. An example of such a variable would be the presence of an infection before the agent has started displaying any symptoms or has been diagnosed by a doctor.
- Whilst the Alpha data model does not explicitly make use of them, they could be **incorporated via the Patient kwargs variable**, which can be set in the config script.
- If common hidden variables were identified, these could be added as Patient attributes, perhaps with some naming convention (such as a prefix hidden_) to clearly indicate its scope. Adding hidden variables and acting on their values will require changes to the core codebase, but should not be too complex to implement (although this depends on their intended use).

P4. Expansion to multi-agent simulations, where multiple patient pathways can be simulated concurrently

Complexity

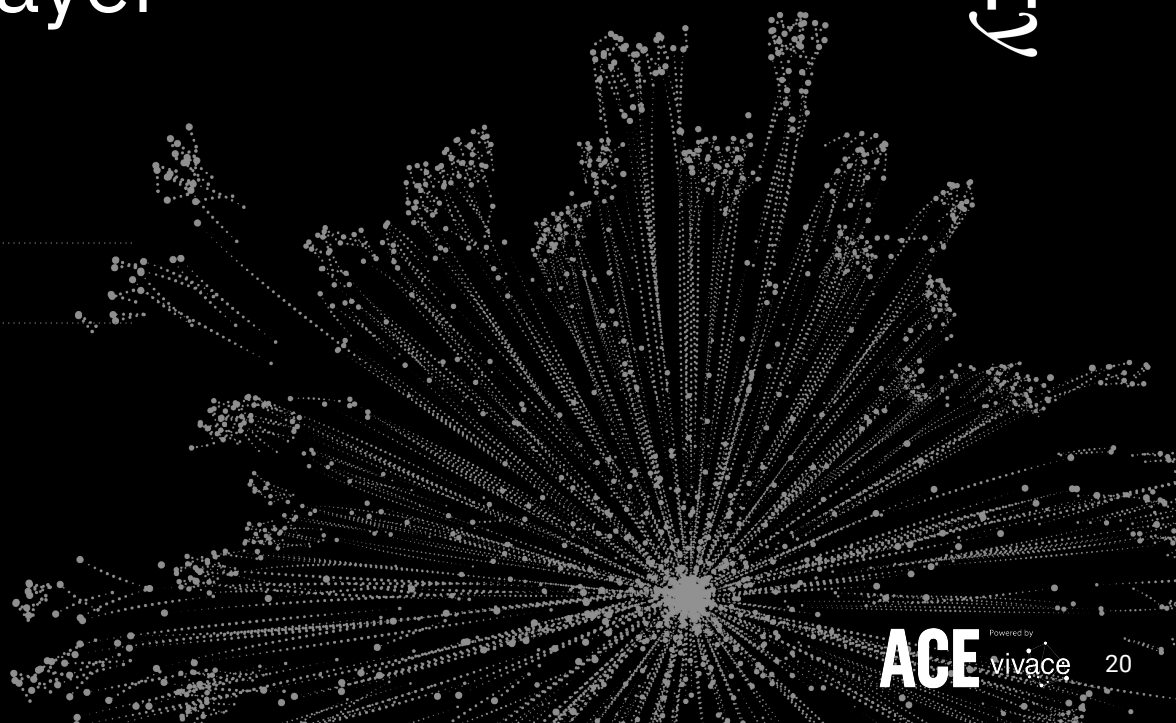
Hard

- The Alpha data model handles multiple patients independently.
- **Whilst it is technically possible to initialise a simulation with many patients in the config.json, the model will simulate patients one after the other in a loop** - all with independent environments and no interactions between patients whatsoever.
- This process should be trivially parallelizable; one small enhancement would be to use a tool like **python's multiprocessing library** to parallelise the simulation.
- A **true multi-patient agent-based simulation would involve significant changes to the codebase**. If the user wishes to go down this route, they should be aware of the following:
    - It might be possible to still have the Intelligence Layer focussing only on updating the record of one central patient at a time - however, it would need to be aware of other patients in its vicinity (e.g. for modelling the likelihood of one patient infecting another with COVID-19). This would require tracking all patient locations and the distances between them.
    - For modelling patients further away (i.e. in other environments that the central patient may need to go to next to inform waiting times / capacity) - this would require the ability to recognise when multiple patients are occupying the same environment, implementing functions to prevent forbidden events such as two unrelated patients having the same GP appointment at the same time
- Depending on the logic required to process and update and patients and environments, **this could be quite a complex enhancement** and it may be more efficient to rewrite the framework.

4. Environment layer

Design features

Recommendations for the future



For the Alpha data model and template language, the Environment incorporates the following attributes and methods

EnvironmentAgent

Attributes (Environment Agent properties)

- `environment_id: Union[str, int]` Unique ID for the environment
- `environment_type: str` String defining environment type, e.g. "hospital". Note this is called "type" in the config.json
- `name: str` Name for the environment, e.g. "Charing Cross Hospital"
- `patient_present: bool` Whether the patient is physically present when interacting with this environment
- `location: Optional[dict]` Location of the environment
- `organization: Optional[dict]` The environment Organization
- `practitioners: Optional[list]` List containing practitioner(s) at this environment
- `interactions: Optional[list]` List of interaction names that the intelligence layer can apply to the patient when visiting this environment
- `patient_data: Optional[DefaultDict]` Store patient data like scans or letters
- `patient_interaction_history: Optional[DefaultDict]` Log of patient interactions, keyed by patient_id
- `capacity: Optional[List[dict]]` Capacity of environment over time
- `wait_time: Optional[List[dict]]` Wait time of the environment over time
- `**kwargs,` Keyword arguments set as attributes

Methods (Environment Agent functions)

- `update` Update patient interaction history (in future, also update capacity and wait_time)
- `save` Save Environment agent as tar file
- `load` Load Environment agent from a tar file

For the Alpha data model and template language, the Environment incorporates the following design features

- **Environment definition:** the Environment represents a physical location (e.g. hospital) or a more abstract entity (e.g. multidisciplinary team meeting). This interacts with the patient record, and results in an update to that record.
- **Environment object:** the Environment object has been designed to not be overly specific in its attributes and methods so that it can be generalised. It enables the identification and creation of more specific Environment subclasses in future which can represent particular interaction points on certain pathways. We have created A&E and GP Environment subclasses as examples, but currently they look almost identical to the parent Environment class.
- **Environment attributes:** the set of attributes attached to the Environment are common to many Environments. This was deduced from numerous interviews with SMEs and research on patient pathways. Attributes such as location and organisation can be used to make the simulation produce more realistic outputs or as part of the Intelligence Layer's decision making process.
- **Role of the Environment:** in the current simulation, the Environment holds a list of interactions that the Intelligence Layer can apply to the Patient, and holds a record of which Patient interacted with it (this list gets automatically populated)
- **Future use of the Environment could involve more updates and dynamics:**
  - E.g. the flag patient_present could help to automate some of the patient record generation
  - E.g. the placeholder Environment attributes capacity and wait_time could be used to affect the times of patient record updates
  - E.g. Data about the patient, such as images and scans, could be stored in patient_data
  - E.g. Location / org / practitioner at an Environment could affect decision making in the Intel layer, and written into the record
  - Environments could "interact" through correlated capacity and wait_time, or accessing each other's patient_data
  - E.g. for large scale pathways with multiple Environments, it may be more optimal for the Intelligence layer to create these

In future, we recommend the following enhancements to the Environment and Intelligence Layers:

	Enhancement	Complexity
E1	Ability to capture / store other interactions / communications beyond the patient agent (e.g. letters)	Medium
E2	Incorporation of dynamics within the Environment layer	Medium

The following slides provide recommendations on how the above enhancements could be implemented.

E1. Ability to capture / store other interactions / communications beyond the patient agent 1/2

Complexity

Medium

- As well as generating a patient health record update, the **Intelligence Layer could produce data to accompany a particular interaction - such as the image of a scan,** or a discharge letter for a patient.
- Currently we have not modelled an interaction function that can produce such data - however, each **environment has an attribute patient_data** that could be used to store such information.
- patient_data is chosen as a defaultdict(list) python type so that the keys can be set as the patient_id, and the values are lists containing the data. We envisaged it could be used as follows:
    - Patient and environment interaction using interaction function - interaction generates some data, for example, a scan
    - The data could look like a dictionary:

```
{
        real_time: 2021-03-24 15:55:25,
        patient_time: 2021-03-24 15:55:25,
        environment_database_name: PACS,
        visible_to_environment_ids: [1, 3, 15],
        patient_record_indices: [22, 23, 24],
        interaction_name: write_letter,
        content_type: image,
        content: <scan.png>,
}
```

    - This would then be **appended to the patient_data[patient_id] list**

E1. Ability to capture / store other interactions / communications beyond the patient agent 2/2

Complexity

Medium

- In the Alpha data model, since only one patient and one environment are passed into an interaction function, an environment can only view its own patient_data information.
- A future version which **gives the interaction function access to all environments** would mean that environments can view each other's data (if visible_to_environment_ids permits it).
- Note - we are assuming that the content of the data (letter or scan) can be represented in some useful abstract manner for this to be possible. If an actual letter or image is required, generating such an output would be far more complex and probably require a separate data model to produce such data at the end of the simulation. Note the following:
    - A truly realistic letter or image may not be useful for guiding a patient along a pathways during the simulation.
    - The images here are messages passed between nodes that simulate what happened in the real world (when a Consultant receives a diagnostic image).
    - Actual images are not being used in the model, so would be an enforced addition with relatively high complexity for low precision.
    - Actual images would require (a) an image generation model which needs to be realistic (b) a component which would be able to interpret the image content and make decisions and (c) a larger amount of memory to store the image. For the purpose of the ABM, only certain attributes of the image are required, such as metadata. We recommend this approach.
- If the ability to model letters was implemented then the ABM could handle more general interactions such as private healthcare referrals, widening the scope to include patient data outside the NHS.

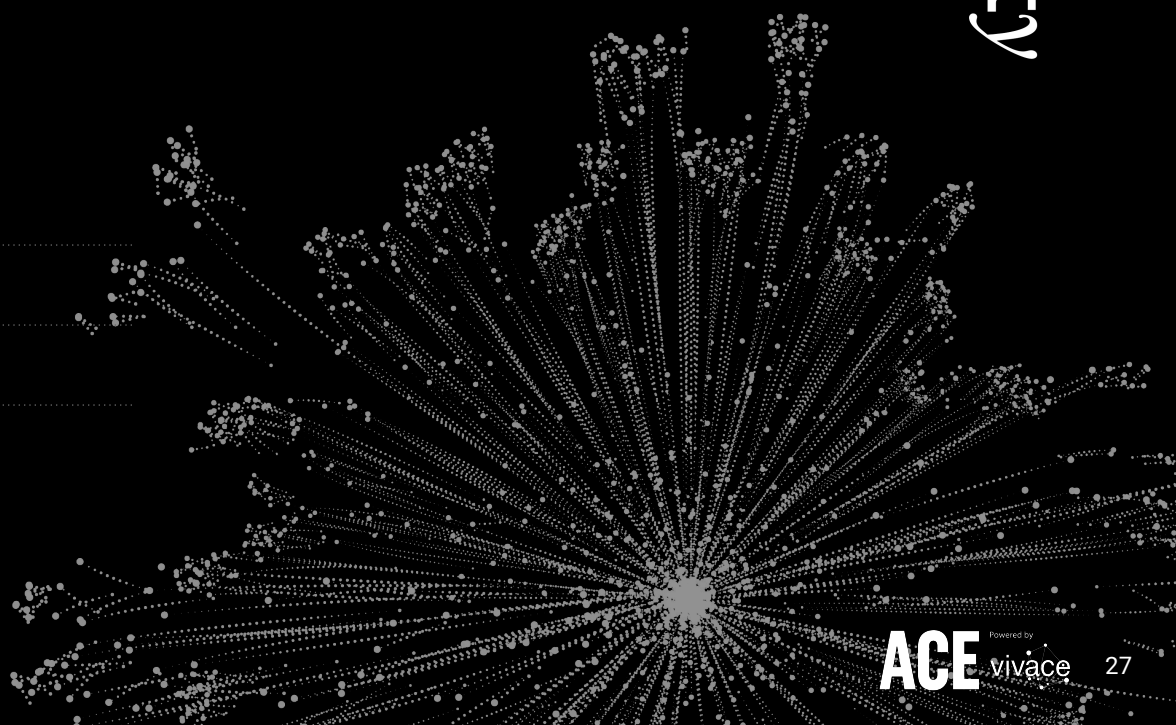
E2. Incorporation of dynamics within the Environment layer

Complexity

Medium

- There are versions of environment dynamics which may not be too onerous to implement.
- For example, consider a GP in-hours service. A simple **rules-based model** could capture:
    - Low capacity / long wait times, between the hours of 8am-10am and 3pm-6pm, Mon-Fri
    - High capacity / short wait times, between the hours of 10am-3pm, Mon-Fri
    - And no capacity otherwise (including bank holidays).
- Each Environment has **capacity and wait_time placeholder attributes** which could be used to store such information, and this could then be used by the interaction function to decide on whether the patient needs to wait an amount of wait_time before creating the first record entry. Also, if the interaction layer has access to other environments, it could use this information to decide on the time of next interaction or to schedule an appointment.
- More complex versions where more patients or other external events are taken into consideration could be employed, but this would require far more engineering.

5. Simulation template language



Overview of the structure of the templating language

Format: Template for the simulation configuration file `config.json`. Angular brackets <...> indicate a name that can be substituted

`patients_attributes:`
List of dictionaries, where each dictionary contains patient __init__ arguments as keys and their corresponding values. Required keys are id, gender, and birth_date. Alternatively, `patients_attributes` can be a path to a JSON (preferred) or CSV containing the equivalent data.

`environments_attributes:`
Same as above but for environments. Required keys are id and interactions, which is a list of interaction functions that can be applied to the patient when it visits that environment (more on this in next slides). Note - any default interactions are automatically added.

`intelligence_dir:`
Path to intelligence layer (more on this in following slides).

`save_dir:`
Directory where outputs (logs, agents, FHIR report) should be saved.

There are numerous other fields that can be set in the config.json, the full specs will be provided in the core codebase.

config.json

```
{
      patients:
      <patients_attributes>,

      environments:
      <environments_attributes>,

      intelligence_dir:
      </path/to/intelligence/dir>,

      save_dir:
      </path/to/save/dir>,

      stopping_condition:
      <stopping_condition>

      ...
}
```

An overview of using the template to run a simulation

- Fuller instructions are provided in the README of the core codebase, which is a python package called patient_abm.
- First clone the repo, cd inside, and install the package via pip install .
- Create the config.json and intelligence directory
- In the terminal, run:

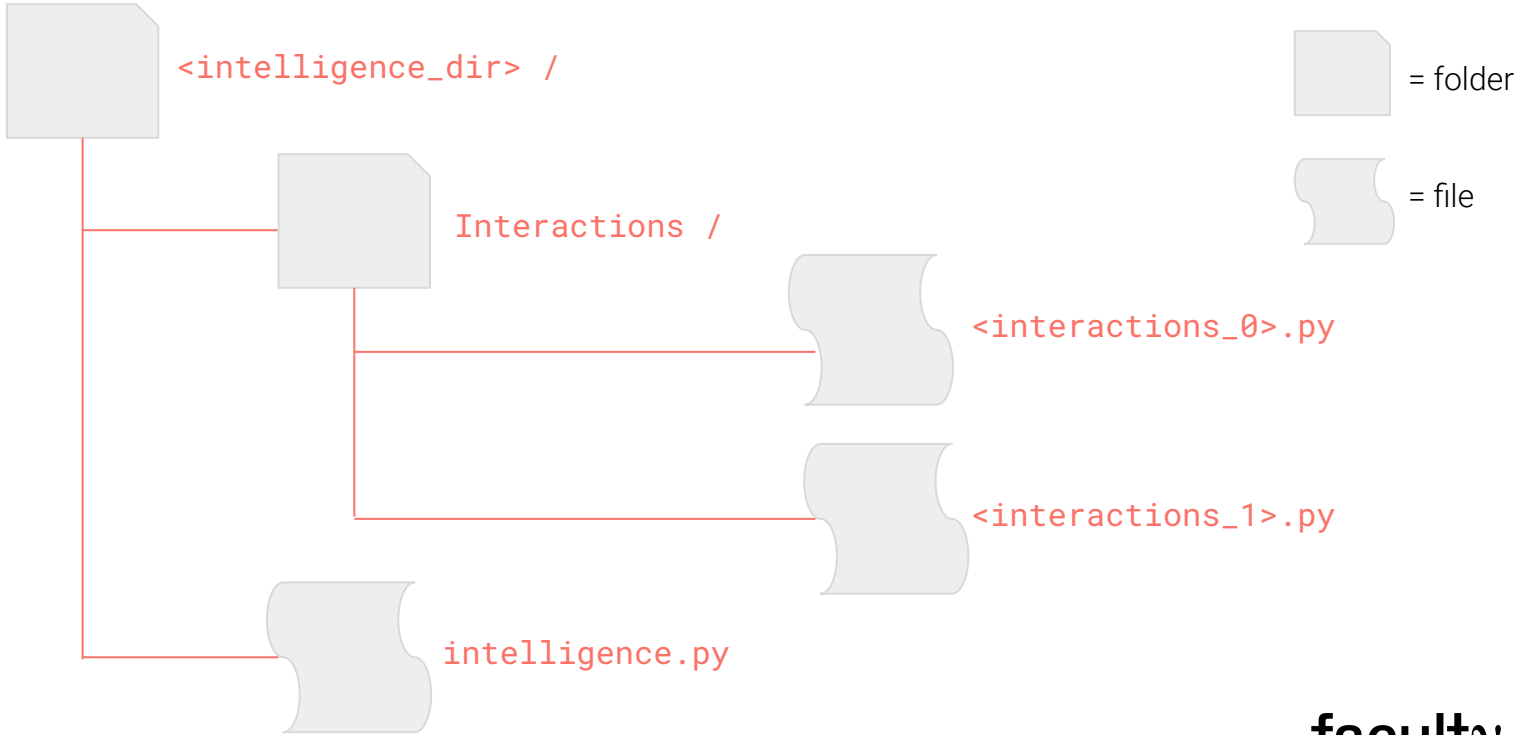
```
patient_abm simulation run --config_path </path/to/config.json>
```

The following folder structure and outputs are created in `save_dir`:

```
<simulation_id> /                                   A new simulation_id is generated for every patient
      agents /
            patient_<patient_id>.tar                The saved patient agent
            environment_<environment_0_id>.tar      The saved first environment agent
            environment_<environment_1_id>.tar      The saved second environment agent
            …
      fhir /
            bundle.json                             The generated patient FHIR health record
      main.log                                      The main simulation log file (JSON lines format)
      patient.log                                   The patient log file (JSON lines format)
```

An overview of the template for the intelligence directory `intelligence_dir`

This directory can be located anywhere.



An overview of the template for the intelligence module `intelligence.py`

This must at least contain a function called intelligence, with a template shown below. The rest of the contents of this module are for the user to define.

```
def intelligence(
      patient,
      environment,
      patient_time,
      interaction_mapper,
):

      < some logic that decides which
      interaction function to apply to
      the patient and environment >

      return (
            patient,
            environment,
            patient_time,
            update_data,
            next_environment_id,
            interaction_name,
            next_environment_id_to_prob,
            next_environment_id_to_time,
      )
```

An overview of the template for the interactions modules <interaction>.py

The names of these modules and the functions they contain are referred to from the environment attribute interactions.

```
def my_interaction(
      Patient,
      Environment,
      Patient_time,
):

      < make new patient record entries,
      calculate probabilities and times
      to next environment. Possible
      custom updates to patient and / or
      Environment (default updates done
      automatically) >

      Return (
            Patient,
            Environment,
            Patient_time,
            next_environment_id_to_prob,
            next_environment_id_to_time,
      )
```

For example, if this script was called: **my_interactions.py**

Then the environment attribute **interactions** would refer to the function **my_interaction** as:

```
interactions = [
      “my_interactions.my_interaction”
]
```

The user is free to name these interactions, modules and functions in anyway they like, as long as they can be found inside the **intelligence_dir** as:

interactions/<module>.<function>

In a future version, to allow more flexibility, some of these naming conventions could be relaxed (e.g. don’t assume a folder called **interactions**) or even design a template that can read from multiple **intelligence_dirs** (with care taken to avoid name clashes)

Design of the simulation template language was informed by the following considerations

- **Follows core design principles of generalisability:** the patient simulation model needed to generate realistic but synthetic electronic health records, and be flexible enough to be further developed to model any patient pathway.
- **Does not rely on fully pre-specifying an entire clinical pathway**: the result of this is core python package patient_abm which handles Patient and Environment agents as objects, as well as controlling the simulation. There is also an Intel layer for which we specify a template language, but otherwise the implementation details are left to the user.
- **Inspired by the https://hash.ai/ framework**: Hash is a platform that provides open source tooling for agent-based modelling, and offers a very similar structure to that required of the simulation template, which could be customised for our specific use case.
- **Outputs a synthetic electronic health record in FHIR v4 format**: chosen because FHIR is a common and well-documented standard for health records. Other formats can also be converted to / from FHIR, plus other standards (such as the PRSB Core Information Standard) are based on FHIR. Also plenty of open source support for FHIR, for example, offline and online validators.
- **Informed by the Synthea library https://github.com/synthetichealth/synthea:** Synthea was useful in providing insight into the structure of FHIR data, and what a FHIR health record might look like for a patient pathway. This also made FHIR a logical format to use because it enabled us to quickly iterate on such a short project.

In future, we recommend the following enhancements to the simulation template - these can be split between the Intelligence Layer and config script:

		Enhancement	Complexity
Intel layer	I1	Intelligence Layer updated to include access to multiple environments	Easy
	I2	Inclusion of SNOMED to set additional Patient conditions	Medium
	I3	Library of generic go-to interactions which can be applied across multiple pathways	Medium
	I4	Background disease measures / probabilities (e.g. based on patient age etc.)	Medium
	I5	Expansion to multiple patient pathways for the same Patient Agent	Hard
Config script	C1	Improve the interface for inputting data on the pathway to make more user friendly	Hard
	C2	Implement ability to convert to other data formats (e.g. PRSB)	Hard
	C3	Implement a private FHIR validator https://www.hl7.org/fhir/validation.html	Hard

The following slides provide recommendations on how the above enhancements could be implemented.

I1. Intelligence Layer updated to include access to multiple environments

Complexity: Easy

- We identified several situations where it might be beneficial for the Intelligence Layer interactions to have access to all the environments at every step of the simulation e.g. one environment might need to access another environment's storage of patient data.
- Another example is in relation to environment dynamics - it may help to know the capacity / wait times of other environments in order to determine the time of the next interaction, or schedule an appointment.
- Our recommendation is to **implement a version of the simulation where in each step the Intelligence Layer receives all environments.** The patient would still be directly interacting on one reference environment only, the other environments only serve to help the interaction take some actions. The patient's health record update will still look like it has only visited a single environment.

I2. Inclusion of SNOMED to set additional Patient conditions

Complexity: Medium

- To fully specify a Patient medical condition, a disease name and code are required, and we recommend using a coding system like SNOMED.
- To do this currently, the user would need to **manually write the relevant SNOMED information** into a Patient record entry inside of an Intelligence Layer interaction function.
- An **alternative to this would be to connect the Intelligence Layer to a SNOMED server**. In this case, the Intelligence Layer could contain functionality to generate a keyword and then use this keyword to query the SNOMED server, the response would then automatically populate the Patient condition name and code fields.
- It is very likely that the response would contain multiple codes for the same keyword (perhaps even codes for some other SNOMED entity types, e.g. procedures). A mechanism for selecting a single code would need to be implemented. This could be achieved by designing a process to filter out irrelevant codes. If there are multiple relevant codes (e.g. multiple codes for “long covid”) a probability distribution could be assigned to the codes and then a single one may be sampled.
- A simpler version of this could be implemented if, say, it was known a priori that only a limited set of SNOMED codes might be required. These could be preloaded from SNOMED and written to, for example, a python dictionary, and this is then made accessible to the Intelligence Layer.
- If a SNOMED query resulted in multiple codes in the response, even though a single code would be selected for updating the patient record, it is still possible to store all these responses. This could be done by adding a patient attribute that tracks all these SNOMED codes (along with the query and other relevant parameters), these codes would then be stored with the patient and accessible at the end of the simulation. The codes could also be written to the logs. This raises a further option where the user could define an “event of interest”, namely, a condition such as “multiple SNOMED codes” that prompts a message to alert the user that this event has occurred during the simulation. This “alert” system would need to be implemented.

I3. Library of generic go-to interactions which can be applied across multiple pathways

Complexity

Medium

- It is very likely that many pathways will have a common set of interactions, for example, booking an appointment, or the chance of the patient developing a cold or flu and then going to the GP and being prescribed some medication.
- To save rewriting code, it would make sense to **develop a bank of common interaction functions** which could be imported across different pathways.
- Some simple common interactions (such as the cold example above) may not be too complex. Some work would be required to **implement a method to refer to these go-to interaction modules inside of the Template Language**, especially if all the scripts containing these features all live in different directories, but this should not be too difficult.
- Developing common functions for something such as all cancer pathways would be a more complex task, but would undoubtedly be a powerful resource.

I4. Background disease measures / probabilities (e.g. based on patient age etc.)

Complexity: Medium

- For more complex and long-term patient pathways, it may be **useful to model the chance that a patient spontaneously develops an illness during the simulation**.
- To make this kind of event realistic, it would be useful to know the **probability of a patient getting ill given their demographics and historical health record**. Determining these probabilities could come from a lookup table, which would have to be made available to the Intelligence Layer. Alternatively a more complex version could be some model which takes in relevant patient features and returns the distribution over illnesses.
- This feature could be enabled and disabled by the user via a config attribute.
- Note a subtlety: patient dynamics are currently fully governed by the Intelligence Layer. This means that a patient state is essentially static until it interacts with its next environment. A patient therefore can only “develop an illness” during an interaction. We could of course model that illness as having started before (by artificially setting an earlier start date) but this kind of dynamic may be limiting. Instead, a **future version of the model could enable patient dynamics even without an environment.**

I5. Expansion to multiple patient pathways for the same Patient Agent

Complexity: Medium

- We have considered the breast cancer pathway in detail and used it to inform the development of the Alpha data model. The breast cancer pathway has helped us shape the structure of all three layers, but in particular it has influenced the attributes of the Environment.
- An issue with the breast cancer pathway is that it is quite linear and very well-specified. There is therefore a risk that either: a) the resulting Alpha data model is not flexible enough to incorporate another pathway, or even multiple pathways at the same time (although we have tested the thinking of our current model on other types of cancer pathway); b) a significant amount of work is required in order to expand the Alpha data model to other patient pathways.
- Throughout the development of the Alpha data model, we have been careful to try and maintain the generalisability of the model. This has been achieved by implementing quite **multi-purpose Patient and Environment objects, which can in future be subclassed to more specific versions**, and **deferring all pathway-specific logic to the Intelligence Layer**, which the user is free to implement. In principle, we therefore believe that no significant changes should be required for the core codebase (which contains the Patient and Environment agents) to work in multiple patient pathway scenarios. Instead, **the complexity would fall into the Intelligence Layer.**
- Nevertheless, there may be certain elements in some pathways that require changes to the core codebase, for instance the social care or mental health pathways may require features that are not yet implemented, such as patient mood - this will require further consideration for how to record such observations (e.g. in python, calling the setattr method on the agent objects - note this is currently used to set patient agent kwargs as attributes)

C1. Improve the interface for inputting data on the pathway to make more user friendly

Complexity: Hard

- The Alpha data model was built to be as pathway-agnostic as possible - we did not want the model to rely on having a pre-defined pathway graph (such as the Synthea approach - https://github.com/synthetichealth/synthea)..
- The pathway manifests itself through the decisions made by the Intelligence Layer. Each interaction calculates a distribution over the environments, this distribution is then sampled and thus the next Environment is chosen.
- One downside to this approach is that if a user does want to apply the simulation to a particular pathway, they would need to be **proficient enough at python in order to write the functionality** to calculate these transition probabilities. This raises a barrier to entry for non-technical users who wish to use the model on a particular pathway.
- A way to ameliorate this issue would be to **design an interface that makes it easy to define a pathway graph** (and possibly the transition probabilities) as part of the model configuration. The graph and probabilities would then need to be parsed by the model and fed into Intelligence Layer.
- An issue would still be that the probabilities of transitioning between environments would change over time, and would depend on the Patient state - these probabilities would be unlikely to equal the initial input probabilities, so the Intelligence Layer would still need to compute the changing distributions (that is, if a realistic and somewhat flexible model was still desired).
- There could be several ways to capture and encode the graphical data:
  - Using a tool like neo4j or networkX
  - Defining an adjacency matrix
  - Each environment could hold a list of “next_environment_ids”*
- Finally, to make the template language generally more robust, a tool like Jinja could be employed.

*currently only next_environment_id provided by the environment interaction is used, the next_environment_id_prob and next_environment_id_to_time are not used.

C2. Implement ability to convert to other data formats (e.g. PRSB) - templating language and in the code repository

Complexity: Hard

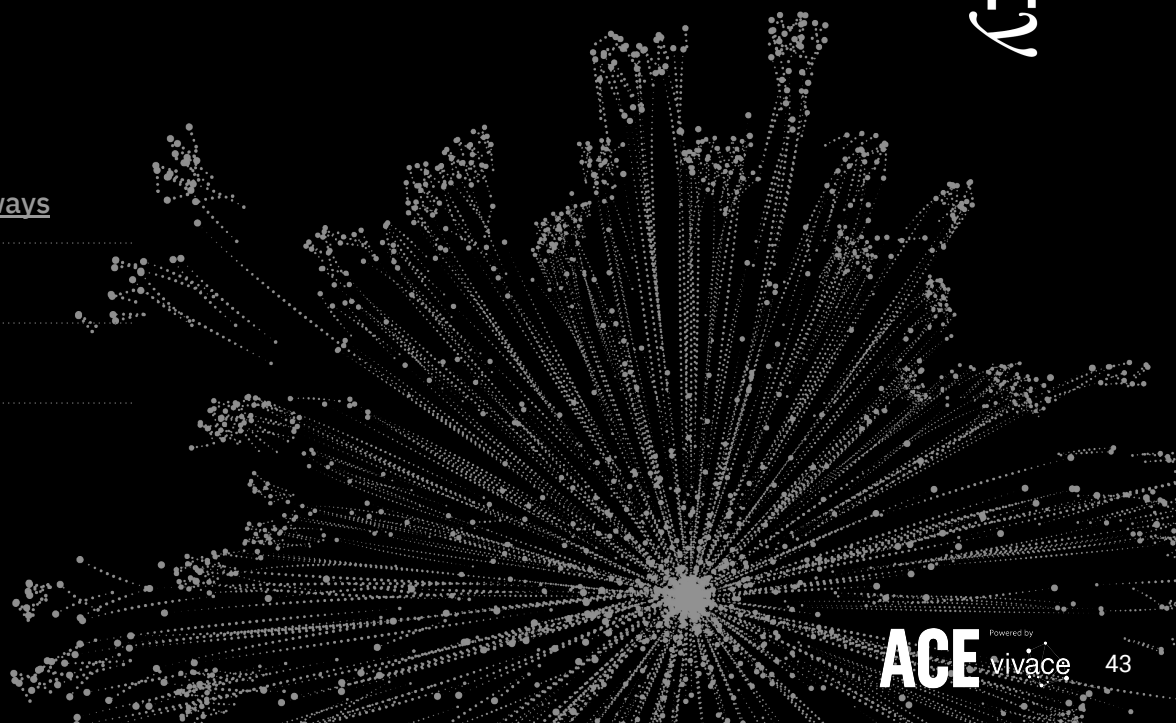
- A key guiding principle for the Alpha data model was to develop a model which laid the foundation for producing realistic patient health records.
- There are numerous formats and architectures for representing electronic health records - e.g. HL7v2, FHIR, art-decor (from PRSB), openEHR. Due to the time constraints in this project, we focussed on generating synthetic patient records in one of these: FHIR (v4).
- There are then two potential ways of converting to other formats:
  a. **Finding or building a data converter which translates from the FHIR v4 to the desired format**. E.g. there seems to be a HL7v2 to FHIR converter (https://github.com/LinuxForHealth/hl7v2-fhir-converter), although we have heard anecdotally that these converters are not very successful. As for PRSB formats, from our understanding, some of the PRSB standards (such as the Core Information Standard) are based on FHIR, hence this conversion may be possible as well (see also enhancement 1 under 'Patient Agent'). There are also tools to translate between different versions of FHIR (e.g. https://www.hl7.org/fhir/r3maps.html)
  b. **Writing a converter directly to / from the internal language we have built.** For user-friendliness, we have developed an "internal patient record representation" which is a simplified language for creating Patient Agent record entries. This saves the user from writing a lot of boilerplate code in the Intelligence Layer and simplifies the logic (for instance, names of fields representing dates dates in different FHIR resource types). To import / export to FHIR v4, we have written a converter which maps from this internal language to / from FHIR. Only a few important fields from the most common resources are currently used, but there is scope to include more depending on the need.
- The converter could be built directly into the core codebase (as it currently is). Or, in a future version, it could be exposed to the user via a template language, which could allow the user to define new maps between fields and FHIR resources.
- These converters should be built in a modular fashion.

C3. Implement a private FHIR validator https://www.hl7.org/fhir/validation.html

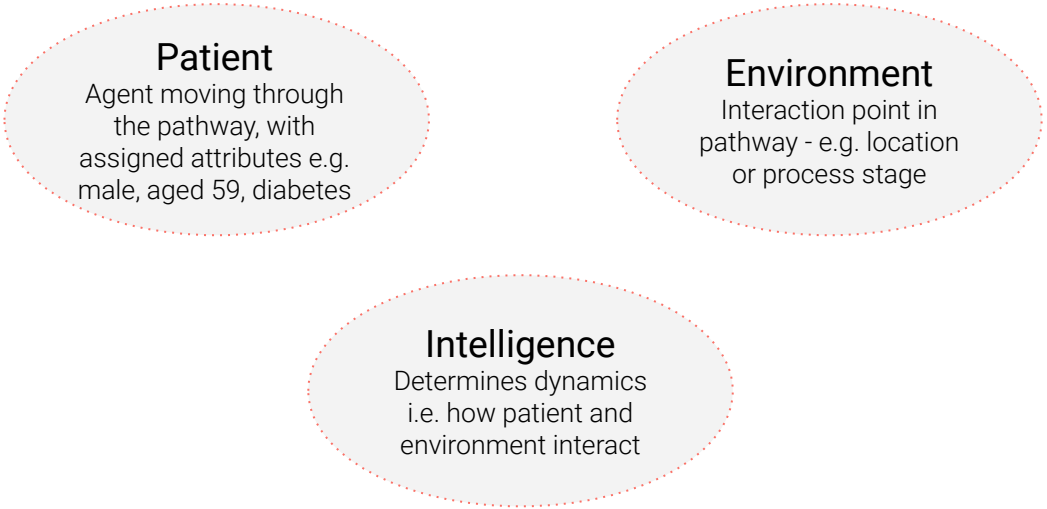
Complexity: Hard

- Validating the FHIR health record that is generated by the Alpha data model can currently be done in two ways.
- **Offline** - using a python library called fhir.resources - https://github.com/nazrulworld/fhir.resources:
  - Pros: it is fast
  - Cons:
    - It is not clear how fast the library keeps up with new FHIR standards
    - Seems to not be rigorous enough - we have found examples that pass offline validation checks but fail online
- **Online** - by connecting to the HAPI FHIR server - https://hapifhir.io/:
  - Pros: listed as an official FHIR test server (https://wiki.hl7.org/Publicly_Available_FHIR_Servers_for_testing), so we expect it is kept up to date with current FHIR standards
  - Cons: it is slow
- To combine the best of both worlds, namely, a fast validator that **keeps up to date with changing FHIR standards**, we recommend implementing a **private FHIR validation server in the future**. This could have the added benefit of building a custom version that, for instance, validates FHIR data that is also compliant with PRSB core information standards.
- We would recommend implementing validation at the point when the config is being read, and when an output is generated. This will protect against an invalid entry / output being generated.

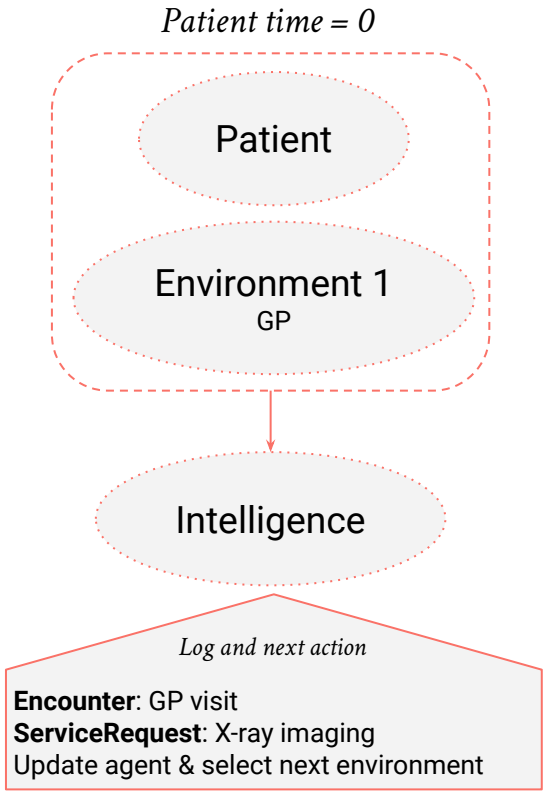
6. Appendices



There are three core ‘layers’ for an agent-based model simulator



At each time step of a simulation, a Patient Agent interacts with an Environment; the first step in the below example simulation is a patient visiting a GP:



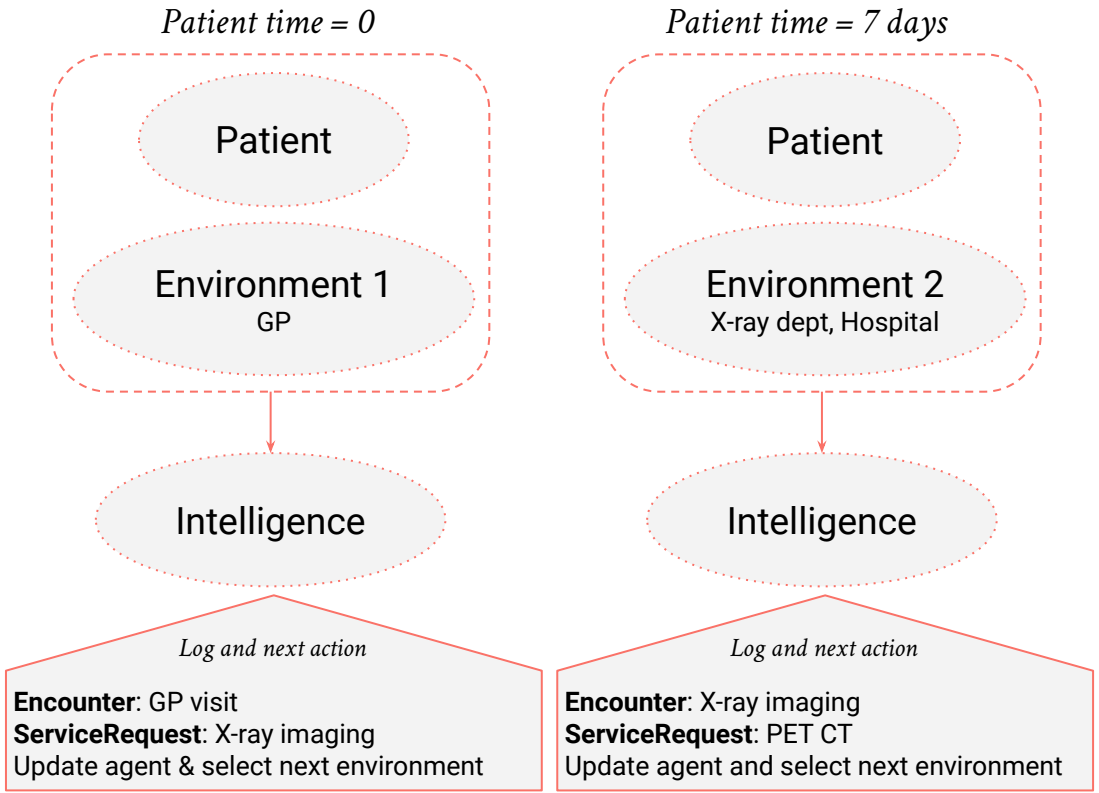
Commentary

The interaction between the Patient Agent and Environment is governed by the **Intelligence Layer**.

The role of the **Intelligence Layer** is to:

- Choose the interaction(s) to apply to the patient and environment and, if relevant, in what order;
- Decide on which Environment the patient should interact with next, and at what "patient time"

The Intelligence Layer uses the Patient Agent and Environment to determine the next action; in this example, this is to have an X-ray at a hospital:

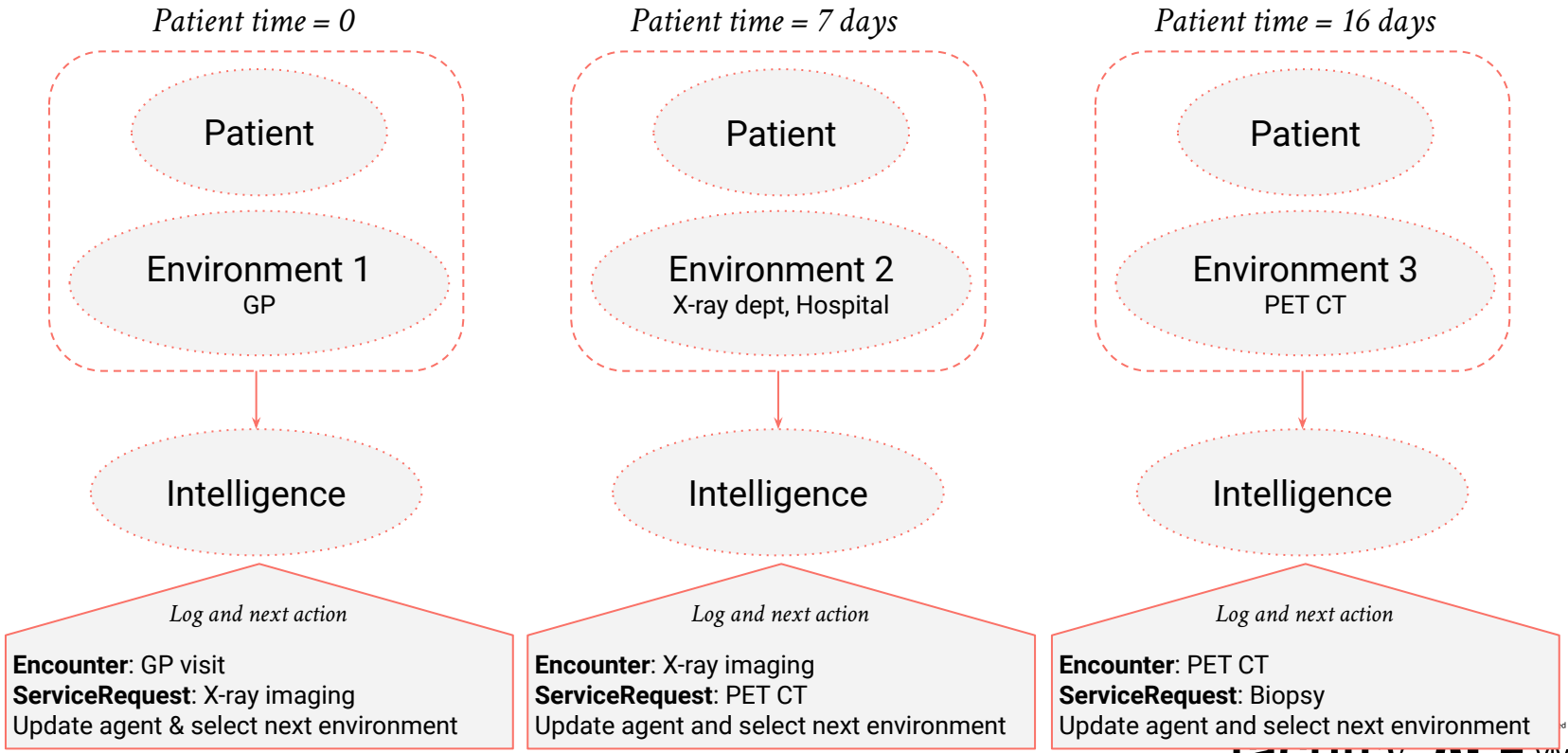


Commentary

- Patient initially visits the GP (at time = 0)
- Intelligence Layer uses currently available information from the Patient, GP and other sources (e.g. rate of tumour formation in a patient with the given demographics)
- Intelligence Layer decides that in the next time step the Patient should go for an X-ray. The visit and outcomes are logged
- In the next time step (patient time = 7 days), the Patient Agent presents itself in the X-ray department at their local hospital and undergoes X-ray
- The Intelligence Layer, which now has the output of the GP visit and the X-ray, decides that the next step should be a PET CT scan in 8 “patient days”

Next slide displays the CT scan interaction.

From the result of the GP visit and X-ray, the Intelligence Layer determines the next step is to have a PET CT scan. The pathway then continues...



The ability to simulate these kinds of patient pathways has a number of potential benefits

- Most synthetic Patient Pathways data is a static representation, and in some instances requires expert clinical input
- It would be useful to have a data source detailing the history of how the data has been created, how it has been passed from system to system, and how it has been impacted by certain changes
- This would allow:
    1. End-to-end system design for architecture and software development
    2. Impact simulations demonstrating the impact of a change / decision on the data flows in a system
- Crucially, the idea behind the abstract data model and templating language is that it is generic enough to create simulations for many different types of patient pathways

Review / comparison of libraries for data model implementation 1/2

Github library	Findings
fhir.resources Python FHIR JSON schema validator	• Written in python, uses pydantic • Easy to use
xmlschema Python XML schema validator	• Written in python • Easy to use
google/simhospital Synthetic FHIR generation	• Written in golang, runs in docker container • HL7v2 text output written to terminal or file • Docs suggest it can generate JSON resources as well, though not had success with this yet • May be possible to convert HL7v2 to FHIR e.g. https://github.com/LinuxForHealth/hl7v2-fhir-converter • Docs suggest can configure “pathways” - requires further investigation
smart-on-fhir/sample-patients Synthetic FHIR generation	• Written in python - required conversion from python 2 to 3 and some debugging • Outputs FHIR in XML format • JSON is preferred so we can use fhir.resources to validate • Tried converting using FHIR resource operation $convert but no luck • Checked XML against FHIR xsd schema using xmlschema - failed to validate Bundle or Patient resource (whereas e.g. https://www.hl7.org/fhir/patient-example.xml.html validates OK) • XML will take some time to parse manually
syntheticheath/synthea Synthetic FHIR generation	• Written in java • Generates FHIR in JSON format • Appears to be US-based • Small sample passes fhir.resources Patient validation • Sanity check: can convert to XML and back using FHIR $convert

Review / comparison of libraries for data model implementation 2/2

Github library	Findings
https://hapifhir.io/	• This is a server which can perform FHIR resource operations • It is easy to connect using python's `requests` library • We have tested to see whether it can validate FHIR data, which it can and so we will use this in the Alpha data model to validate the simulation output, in agreement with what was discussed this week
https://hash.ai/ Open-core platform for creating agent simulations	• An online platform for developing ABMs • Most examples are in javascript • It provides good ideas and sources of inspiration for the setup, and in particular the intelligence layer and template language

Useful resources 1/2

Technical resources:

https://healthdatainsight.org.uk/project/the-simulacrum/
https://pypi.org/project/fhir.resources/
https://github.com/smart-on-fhir/sample-patients
https://docs.smarthealthit.org/
https://github.com/google/simhospital
https://hapifhir.io/
https://github.com/synthetichealth/synthea
https://hash.ai/
https://art-decor.org/art-decor/decor-project--prsb03-
https://fhir.hl7.org.uk/StructureDefinition
https://simplifier.net/ukcore/~resources?category=Profile&sortBy=RankScore_desc
https://theprsb.org/standards/coreinformationstandard/
https://medium.com/neo4j/modeling-patient-journeys-with-neo4j-d0785fbbf5a22
https://github.com/Neo4jSolutions/patient-journey-model
https://www.openehr.org/
https://jinja.palletsprojects.com/en/2.11.x/
http://jasss.soc.surrey.ac.uk/14/2/5.html

Useful resources 2/2

Patient pathway resources:

https://www.england.nhs.uk/wp-content/uploads/2019/10/crs-progress-report-v5-311019.pdf

https://www.nuffieldtrust.org.uk/news-item/a-new-era-for-a-e-targets-what-will-be-the-impact-of-the-new-basket-of-measures#the-whole-basket

https://www.health.org.uk/publications/long-reads/returning-nhs-waiting-times-to-18-weeks

http://www.cancerservicesdirectory.wales.nhs.uk/sitesplus/documents/1112/Breast%20NOP%200bFINAL.pdf

https://pathways.nice.org.uk/pathways/advanced-breast-cancer#content=view-node%3Anodes-imaging-assessment&path=view%3A/pathways/advanced-breast-cancer/advanced-breast-cancer-overview.xml

https://www.northerncanceralliance.nhs.uk/pathway/pathway-service-improvement/cancer-pathways/

https://www.cancer.org.au/assets/pdf/acute-myeloid-leukaemia-optimal-cancer-care-pathway