同一套解析与中间表示（IR），一轨走“解释执行”（runtime 直接跑 XML），另一轨走“代码生成”（Jinja2 出可审计 Python/SimPy 源码与单测）。整个流程以你给的 enhanced-battlefield-simulation-fixedV2.3.xml 与规范 eati\_schema\_v2\_2\_1\_complete.xml 为基准。

**1) 总体架构与目录**

eati-sim/

├─ eati\_core/ # 解析/验证/IR/表达式/分布/调度的“内核”，双轨共用

│ ├─ parsing/ # XML→AST→IR

│ ├─ validation/ # XSD + 语义校验

│ ├─ ir/ # dataclasses: Model, Entity, Action, Activity, Interaction...

│ ├─ mapping/ # 分布/表达式/资源/交互 映射表（XML枚举→运行时/模板）

│ ├─ expr/ # 表达式求值与沙箱

│ ├─ dist/ # 各类随机分布

│ ├─ runtime/ # 解释执行器（SimPy集成）

│ ├─ bus/ # 交互/消息总线（sync/async）

│ └─ monitor/ # 采样、指标、导出

├─ tracks/

│ ├─ interpreter/ # 解释执行轨：装配 IR→SimPy 进程与资源

│ └─ codegen/ # 代码生成轨：Jinja2 模板、渲染器、生成源码与单测

│ ├─ templates/

│ │ ├─ package/\_\_init\_\_.py.j2

│ │ ├─ main.py.j2

│ │ ├─ env.py.j2

│ │ ├─ entities.py.j2

│ │ ├─ resources.py.j2

│ │ ├─ actions.py.j2

│ │ ├─ activities.py.j2

│ │ ├─ interactions.py.j2

│ │ ├─ monitors.py.j2

│ │ └─ tests/test\_sim.py.j2

│ └─ renderer.py

├─ examples/

│ ├─ enhanced-battlefield-simulation-fixedV2.3.xml

│ └─ run\_interpreter.py # 解释执行轨端到端样例

├─ tests/ # 回归测试（双轨都跑），对齐断言

└─ cli.py # CLI：validate/run/gen

**2) XML→IR：解析与验证（双轨共用）**

* **XML 校验**：先用 XSD (eati\_schema\_v2\_2\_1\_complete.xml) 做结构校验，再做**语义校验**（引用完整性、ID 唯一性、条件/公式所需变量存在、时间分布参数齐全、交互 Source/Target/Activity/Action 对应等）。
* **抽象语法树（AST）**：按模块切分（ModelMetadata/SimulationConfig/GlobalVariables/Resources/Entities/Actions/Activities/Interactions/Events/Monitors/ExternalInterfaces）。
* **IR 数据模型**（Python @dataclass 简要示意）：
* @dataclass
* class SimulationConfig: end\_time: float; run\_mode: str; seed: int; step\_size: float; ...
* @dataclass
* class DistSpec: kind: str; params: dict; trunc: tuple|None
* @dataclass
* class Expression: formula: str; language: str="python"; return\_type: str|None=None
* @dataclass
* class Activity: id: str; name: str; kind: str; delay: DistSpec|None; inputs: dict; outputs: dict; impl: Any
* @dataclass
* class Action: id: str; name: str; activities: list[str]; trigger: Expression|None; start\_conds: list[Expression]
* @dataclass
* class Resource: id: str; name: str; type: str; capacity: int; init\_qty: int|None; monitor: dict
* @dataclass
* class Interaction: id: str; name: str; type: str; protocol: str; source: str; target: str; source\_act: str; target\_act: str; semantics: dict
* @dataclass
* class Entity: id: str; name: str; attrs: dict; state: dict[str,Any]; actions: list[str]; handlers: list[dict]
* @dataclass
* class Model: conf: SimulationConfig; globals: dict; resources: dict[str,Resource]; activities: dict[str,Activity]; actions: dict[str,Action]; entities: dict[str,Entity]; interactions: dict[str,Interaction]; events: ...; monitors: ...
* **表达式/路径解析**：把 XML 中的 global.X / self.Y / context.Z / message.\* / res\_\* / env.now 解析为 IR 中的**取值路径**与**访问器**，供解释器与模板共享。

**3) 分布/表达式映射表（双轨共用）**

* **分布映射**（XML→运行时策略/模板宏）：

| **XML Distribution** | **运行时采样函数（解释）** | **模板宏（代码生成）** |
| --- | --- | --- |
| constant(value) | lambda p: p["value"] | J2: sample\_constant(value) |
| uniform(min,max) | random.uniform(min,max)（可截断） | J2: sample\_uniform(min,max,trunc) |
| normal(mu,sigma) | np.random.normal(mu,sigma)（可截断） | J2: sample\_normal(mu,sigma,trunc) |
| gamma(shape,scale) | np.random.gamma(shape,scale) | J2: sample\_gamma(shape,scale) |
| …（与 XSD 列举同步） |  |  |

* **表达式映射**：
  + 解释轨：安全沙箱 eval（限制内建 + 只暴露 env, global, self, context, message, res\_\* 的只读代理），可选 AST 白名单。
  + 代码轨：渲染到函数体/条件语句；对 global.X / self.Y 等用**模板过滤器**注入正确变量名与作用域。

**4) 验证器（结构 + 语义）**

* **结构**：按 XSD 过。
* **语义**（关键规则）：
  + 引用存在性：ActionRef/ActivityRef/ResourceRef/InteractionRef/EntityRef 均需可解。
  + 条件类型一致：ReturnType==boolean 的表达式只能放在条件位。
  + 分布参数齐全、数值范围符合（如 Probability ∈[0,1]）。
  + 交互语义闭环：source\_act/target\_act 存在，协议匹配（sync 必须有 ResponseHandling）。
  + 资源/容器：初始数量≤容量；队列策略合法。
  + 监控/导出：采样周期>0，目标存在。
  + 事件/任务：触发与完成条件互不矛盾。

验证通过后产生签名（hash），写入运行与生成产物，便于回归与溯源。

**5) 解释执行轨（Interpreter Track）**

* **调度器**：SimPy Environment；按 RunMode 支持 continuous/step/debug。把 WarmupTime/StepSize/RealTimeRatio 做适配。
* **资源建模**：根据 Resource.type 映射到 simpy.Resource / PriorityResource / Container / Store 并封装监控钩子（利用 request()/put()/get() 事件记录利用率/队长）。
* **实体与动作**：
  + 为每个 Entity 构造一个“代理进程”，内部根据其 Actions 和 Trigger/StartConditions 生成**进程网络**（顺序/并行由 Action.activities@executionMode 决定）。
  + Activity.kind 分三类：
    1. internal/interactive/compound：由解释器内置执行（计算/赋值/决策/子活动编排），按 DelayTime yield env.timeout(...)。
    2. simpy：直接把 XML 提供的 ProcessOperation/Code 封装成协程执行（沙箱执行，传入限定上下文）。
    3. external：通过**适配器**调用外部模块（如 combat\_algorithms.execute\_artillery\_barrage），带错误策略（log\_and\_continue / raise / retry）。
* **交互/消息总线**：
  + async publish-subscribe：在总线上以主题/队列实现；QoS/Ordering 用有界队列 + 序号模拟；DeliveryGuarantee（at-least/exactly-once）通过 ack/重投实现。
  + sync request-reply：源进程 yield 等待响应事件；目标侧按 ReceiveConditions 检查后触发 TargetAction/Activity 并组织响应（可延迟）。
* **事件与监控**：
  + 事件（Event）在独立进程中轮询条件或定时触发。
  + 监控器（Monitor）按 SampleInterval 抽样并导出 json/csv，可通过 WebSocket 推送（若 InterfaceConfig.enabled）。

**6) 代码生成轨（Codegen Track）**

* **输入**：同一份 IR + 同一套映射表。
* **模板切分**：与解释轨模块边界一致（env/resources/entities/actions/activities/interactions/monitors），确保生成代码结构与解释执行期的一致性（路径、命名、一致的接口）。
* **生成产物**：
  + package/：可直接 pip install -e . 或 python -m package.main 运行的**可审计源码**。
  + tests/：根据 IR 自动生成**最小单测**与**端到端回归用例**（对齐解释轨的断言，见第 8 节）。
  + README.md：从 ModelMetadata 与 SimulationConfig 渲染文档（可选）。
* **模板要点**：
  + 延迟与分布使用模板宏（与解释轨共享同名实现，保证行为一致）。
  + 表达式位（条件/赋值/上下文映射）通过过滤器把路径替换为生成代码中的同名变量。
  + 交互：生成一个与解释轨总线等价的轻量实现（或直接依赖同一内核 eati\_core.bus，保证一致）。

**7) 统一的执行契约（两轨同接口）**

* 约定同一入口签名：
* def run(model: Model, \*, end\_time: float|None=None, seed: int|None=None) -> RunResult
* class RunResult:
* logs: list[str]
* metrics: dict[str, Any] # 关键指标（DamageAssessment、rounds\_fired...）
* timelines: dict[str, list] # 事件/状态时间线
* artifacts: list[Path] # 导出的json/csv
* 解释轨：tracks.interpreter.run(model, ...)
* 代码轨：tracks.codegen.renderer.render(model) -> /out/pkg，随后 python -m out.pkg.main --end-time ... 返回同结构结果；或者在同进程内 import out.pkg as sim; sim.run(...)。

**8) 端到端样例与回归测试（以 battlefield XML 为例）**

* **端到端样例（解释轨）**：
  1. parse+validate XML→IR（打印出 IR 摘要与签名）。
  2. interpreter.run(IR, end\_time=1800, seed=123)。
  3. 断言：
     + MissionStatus 最终为 "completed" 或 "ongoing"（由随机数影响，允许区间）。
     + 发生至少一次 int\_enemy\_report → act\_process\_intel → int\_fire\_order → act\_execute\_fire\_mission 的链路。
     + res\_artillery\_rounds.level 减少且 rounds\_fired 增加。
     + global.DamageAssessment ∈[0,1]，大于等于 0.6 出现 BDA 流转。
* **代码轨生成并运行**：
  1. codegen.render(IR) -> out/pkg
  2. 运行 out/pkg/tests/test\_sim.py：
     + 与解释轨同样的断言集合（阈值/范围一致）。
     + 样例中关键时间点（如 StrikeCompleted==True）出现顺序一致（允许 ±Δt 容差）。
* **回归对齐**：tests/test\_parity.py 比较两轨的关键指标（如 BDA 分布均值、事件序列的布尔模式），设定可接受误差与信号灯。

**9) 关键实现要点（难点清单与解法）**

* **表达式安全执行**：解释轨用 AST 过滤 + 受限字典；代码轨只做字面量渲染，不把用户任意代码注入模板。
* **并行与中断**：Action.activities@executionMode 与 act\_cease\_fire 这类 Type=interrupt，统一由**动作执行器**处理（内部维护可中断 Process，接收优先级中断信号）。
* **交互一致性**：DeliveryGuarantee/Ordering/QoS 在解释轨总线上实现；代码轨直接复用同一总线库，避免行为漂移。
* **分布截断**：两轨同一算法（接受 TruncationMin/Max），以重采样或逆变换法实现，保证统计一致性。
* **资源度量**：对 Container/Resource 封装监控探针（enter/leave/put/get 时记录），两轨一致。
* **外部算法**：提供同名 Python 接口 combat\_algorithms.execute\_artillery\_barrage(...)；若缺实现，生成桩函数并在解释轨也用同桩，确保可跑通。

**10) CLI 与使用流程**

* eati validate examples/enhanced-battlefield-simulation-fixedV2.3.xml
* eati run --engine=interp --xml examples/...xml
* eati gen --xml examples/...xml --out build/pkg && python -m build.pkg.main
* eati test --all（跑解释轨、生成轨的端到端与回归对齐）

**11) 可交付物清单**

* **eati\_core** 内核库（解析/IR/映射/表达式/分布/总线/监控）。
* **interpreter 引擎**（SimPy 解释执行）。
* **codegen 渲染器 + Jinja2 模板集**（产出源码与单测）。
* **端到端样例**（用 battlefield XML 跑通两轨）。
* **回归与一致性测试**（tests/）。
* **文档**（架构说明、数据字典、二次开发指引）。

好的，在前面“双轨”（解释执行轨 + 代码生成轨）的基础蓝图上，我继续把方案B补齐到“可以落地”的粒度。下面按“模块 → 数据流 → 关键算法 → 工具链与保障 → 里程碑”来补充。

# 顶层结构（Python）

eati\_runtime/ # 解释执行轨(Interpreter)

├─ parser/ # XML→AST/IR

├─ validator/ # XSD校验、语义/一致性校验

├─ ir/ # 统一IR定义（面向解释器与生成器共享）

├─ scheduler/ # SimPy调度/资源/事件封装

├─ exec/ # 实体/动作/交互/事件解释执行器

├─ expr/ # 表达式求值(安全沙箱)

├─ dist/ # 分布映射(TimeDistribution/随机源)

├─ monitor/ # 监控指标/采样/导出

├─ api/ # Python API与CLI

└─ tests/

eati\_codegen/ # 代码生成轨(Generator)

├─ templates/ # Jinja2模板(包级/模块级/测试/文档)

├─ render/ # IR→模板上下文构造与渲染

├─ post/ # 生成后处理(black/ruff/导入修正)

└─ tests/

samples/

├─ enhanced-battlefield-simulation-fixedV2.3.xml

└─ regression/ # 回归测试快照与生成产物

# 1) 解析与IR（统一服务两条轨道）

## 1.1 XML→AST→IR

* **XSD结构校验**：用 lxml.etree.XMLSchema 做结构校验，先阻断无效模型。
* **AST抽取**：按顶层段落分解（ModelMetadata/SimulationConfig/GlobalVariables/Resources/Entities/Actions/Activities/Interactions/Events/Monitors…），保留原始位置信息（行号、XPath）便于诊断。
* **IR归一**（关键）：定义面向执行的中立IR（Python dataclasses），让解释器与生成器共享同一份中立模型。
  + IR.SimConfig（时间单位、EndTime、随机种子、模式…）
  + IR.Resource（类型、容量、抢占策略、监控开关）
  + IR.Entity（属性、状态变量、生命周期、可视化）
  + IR.Action（触发、前/终止/中断条件、资源需求、活动序列、优先级/超时/重试）
  + IR.Activity（内部函数/外部算法/延迟分布/SimPy函数映射/子活动）
  + IR.Interaction（源/目标、消息构造、QoS、执行语义、错误处理）
  + IR.Event（触发时间/条件、动作集合、是否重复）
  + IR.Expressions（公式/语言/引用变量）
  + IR.Distribution（类型、参数、截断）
  + IR.InitialActionsConfig（多起始行动执行模式、依赖、优先级/阶段等）
* **表达式与分布提前绑定**：
  + 将 ExpressionType 编译为“可调用闭包”（解释器使用）与“模板字符串片段”（生成器使用），都基于统一的 **符号表**（全局变量、实体状态、上下文数据、监控指标）。
  + 将 TimeDistributionType 预编译为 callable rng()（解释器使用）与“Jinja2宏调用”（生成器使用）。

## 1.2 语义/一致性校验（早失败）

* **引用完整性**：ActionRef/ActivityRef/ResourceRef/EntityRef/InteractionRef 全覆盖校验。
* **初始行动编排**：sequential/parallel/conditional/priority/staged 语义约束与循环依赖检测。
* **资源需求可满足性**：容量/替代资源/抢占策略的静态可行性检查。
* **交互契约**：消息映射字段存在性、验证表达式闭包可编译、超时/重试参数合法。
* **监控定义**：采样窗口、聚合、导出配置的组合合理性。

# 2) 解释执行轨（Interpreter Runtime）

## 2.1 SimPy适配与运行时

* **Env封装**：RuntimeEnv = simpy.Environment + 时钟换算（TimeUnit）、RealTimeRatio（可选）。
* **资源适配器**：ResourceFactory 支持 Resource/PriorityResource/PreemptiveResource/Container/Store/FilterStore 等；队列策略、抢占策略参数化。
* **进程模型**：
  + ActionProcess(entity, action)：根据 ActionActivities 的执行模式（串/并/选择/循环）展开子进程；资源申请→执行活动→释放→错误/中断处理。
  + ActivityNode：支持三类执行单元：内部表达式、外部算法、SimPy原语（如 yield env.timeout(...)、Store.get() 等）。
  + InteractionProcess：封装发送/接收/超时/重试，支持QoS与回调触发。
  + EventScheduler：按触发时间/条件启动动作集，支持重复事件。
* **表达式求值**：
  + 安全沙箱（ast.parse 白名单、禁内建、只暴露数学库、随机库受控接口、只读上下文）。
  + 提供 eval\_expr(expr, context)；上下文含：env.now、全局/实体状态、上一处理结果、监控快照。
* **监控与可观测性**：
  + 资源利用、队长、等待时长；实体/动作耗时；自定义指标。
  + 导出：CSV/Parquet（批量/流式）、可选WebSocket/REST推送（若InterfaceConfig启用）。
* **调试/断点**：
  + 支持 Breakpoints：命中条件触发暂停（step模式）、日志与外部通知。

## 2.2 初始行动与任务生命周期

* 按 InitialActionsConfig.ExecutionMode 驱动启动序列；提供全局/局部超时；失败策略（重试/降级/补偿行动）。
* TaskLifecycle：Start/Completion/Failure/Timeout 条件在运行中持续监控，触发状态迁移与善后。

# 3) 代码生成轨（Jinja2 模板）

## 3.1 模板分层

* **包骨架**：setup.py/pyproject.toml、\_\_init\_\_.py、runtime.py（轻度依赖解释器通用部件以减少重复）。
* **模型到源码**：
  + model\_resources.j2.py：生成资源类与初始构造。
  + entities.j2.py：为每个实体生成 class <EntityName>，包含状态、行为入口。
  + actions.j2.py：将 Action 转成 def proc\_<action>(env, entity, ctx):，活动展开成流程化 yield 序列。
  + interactions.j2.py：生成消息构造/校验/超时重试包装函数。
  + events.j2.py：生成计划任务/条件触发器注册代码。
  + main.j2.py：拼装 SimConfig、实例化资源与实体、注册初始行动、run。
* **表达式与分布映射**：
  + Jinja2 过滤器/宏：|expr(py\_ctx)、dist('normal', mu, sigma, trunc\_min=None, trunc\_max=None) → 渲染成 rng.normalvariate(...) 或 random.expovariate(...) 的可执行代码。
* **单测与回归**：
  + tests/test\_smoke.j2.py：End-to-end 运行不抛异常，关键指标断言。
  + tests/test\_contracts.j2.py：引用完整性、配置一致性断言。
  + 生成后自动 black/ruff 格式化与静态检查。

## 3.2 生成后处理

* **导入修正**：按是否启用外部接口/监控决定可选依赖导入。
* **可审计性**：在源码头部写入生成指纹（模型hash、时间戳、Schema版本），便于审计与追溯。
* **快照**：把生成的源码与运行输出（关键统计）入库samples/regression/<model\_id>/…。

# 4) 双轨协同与切换策略

| **场景** | **解释执行轨** | **代码生成轨** |
| --- | --- | --- |
| 快速迭代/调试 | ✅（热加载XML，定位到XPath） | ➖（需再生成） |
| 性能/可部署 | ➖（有解释开销） | ✅（纯Python源码，轻依赖） |
| 审计/归档 | ✅（运行日志+IR快照） | ✅（源码可读+单测） |
| 外部集成 | ✅（Runtime API/WS/REST） | ✅（嵌入式包/函数接口） |

* **一致性来源**：两条轨共享 **同一 IR 与分布/表达式映射表**，并共用 **语义/一致性校验**。
* **金样核对**：用同一输入随机种子，在解释器和生成源码的程序上跑“关键KPI对齐测试”（均值、方差、事件计数区间）。

# 5) 关键算法与要点

## 5.1 活动编排器（Action→Activities）

* executionMode=sequential：线性 yield 链。
* parallel：为子Activity创建并发进程，汇聚simpy.events.AllOf。
* conditional/choice：表达式驱动分支，支持概率权重。
* loop（SubActivities带LoopCondition）：循环至条件失败或终止信号。

## 5.2 资源与抢占

* PriorityResource/PreemptiveResource：将 Action.priority 与 ResourceRequirements.priority 合并为请求权重；中断时进入 OnInterrupt 分支或重试策略。

## 5.3 交互语义（发送/接收/超时/重试）

* 统一 send\_message()/await\_response() 适配：队列/WS/REST不同通道作为后端插件；QoS(持久化/顺序/最大时延)落到重试与超时参数化。

## 5.4 表达式安全执行

* AST白名单（Num/BinOp/BoolOp/Compare/IfExp/Call(白名单函数) 等）；禁用 \_\_import\_\_、文件/网络/反射；只提供 math, random, statistics(可选)；上下文变量定义为只读 MappingProxyType。

# 6) 工具链与开发保障

* **静态分析**：ruff/mypy（IR类型严谨，模板渲染上下文有类型提示）。
* **持续集成**：pytest + coverage；GitHub Actions（或本地CI）。
* **基准测试**：pytest-benchmark 对“解释 vs 生成”同场景比测。
* **文档**：mkdocs 生成《Schema→IR→Runtime/Generator 映射文档》，自动从代码注释与模板注释抽取示例。
* **CLI**：
  + eati sim run <model.xml> [--mode=interp|codegen] [--seed=…]
  + eati sim gen <model.xml> -o out/ --with-tests
  + eati sim diff --gold <gold.json> --new <new.json>（回归差异）

# 7) 端到端样例与回归（以 enhanced-battlefield-simulation-fixedV2.3.xml 为金样）

1. **校验**：XSD通过；语义校验无误；生成 IR.dump.json。
2. **解释执行**：跑到 EndTime；导出 metrics\_interp.json。
3. **代码生成**：输出可运行包 gen\_sim/ 与 tests/；pytest -q 通过；跑出 metrics\_codegen.json。
4. **对齐**：compare(metrics\_interp.json, metrics\_codegen.json) 达到容差（均值±ε、P95区间重叠等）；若不一致，回溯到映射表/表达式沙箱差异。
5. **基准**：记录两轨性能与内存占用作为基线。

# 8) 风险与对策（补充）

* **表达式差异**（运行时eval 与 生成代码间轻微数值差）：  
  → 统一随机源与数学库；统一舍入策略；模板里显式导入同名函数表（如 rand\_exp(rate) 宏）。
* **并发/中断边界**：  
  → 为 PreemptiveResource 写独立单元测试覆盖“占用/中断/恢复/重试”序列；语义以IR为准。
* **外部算法副作用**：  
  → 解释轨：在隔离线程/进程池执行，设置超时；生成轨：模板生成同样的超时封装与日志。
* **接口可选依赖膨胀**：  
  → 插件化后端（MQ/WS/REST），未启用不导入；代码生成时按IR裁剪导入。

# 9) 里程碑（按可交付物）

* **M1：IR与校验**（第1周）  
  统一IR、引用与语义校验、分布与表达式映射表v1、解析 enhanced-battlefield… 成功导出 IR.dump.json
* **M2：解释执行最小闭环**（第2周）  
  资源/动作/活动/事件/交互最小子集跑通；产出 metrics\_interp.json
* **M3：Jinja2代码生成最小闭环**（第3周）  
  生成 gen\_sim/ 可运行；单测模板通过；产出 metrics\_codegen.json
* **M4：双轨一致性与基准**（第4周）  
  KPI对齐与性能报告；断点/调试/监控完善
* **M5：接口与可视化（可选）**（第5周）  
  WS/REST/数据导出与仪表盘样例；文档与演示