```
ELEMENTO: "have↔need-bridge@s1 (Bidirectional Gap Synthesizer)"
alias: "bridgecraft-bidir"

contexto: {
  proposito: "Detectar la brecha entre lo disponible (HAVE) y lo requerido (NEED), proponer el puente mínimo válido y ejecutarlo en dos modos complementarios: HAVE→NEED (síntesis por adaptación) y NEED→HAVE (síntesis por generación).",
  definiciones: {
    HAVE: "artefactos existentes (datos, código, análisis, dumps, conocimiento implícito)",
    NEED: "contrato/forma/resultado requerido por el pipeline aguas abajo (spec, schema, interfaz)",
    puente_minimo: "secuencia más corta de transformaciones que satisface el contrato sin violar invariantes"
  },
  dualidad: {
    yin: "HAVE→NEED (adaptar/estructurar/normalizar)",
    yang: "NEED→HAVE (materializar/generar instancias que cumplen el contrato)"
  },
  entradas: ["have{}","need{}","catalogo_modular","politica_confianza"],
  salidas: ["plan_conexion","artefacto_resultante","contratos","provenance","metrica_confianza"]
}

orchestrator: {
  name: "have-need.orchestrator",
  describe_how_nodes_interact: [
    "N0.modelar: normaliza HAVE y NEED como contratos comparables.",
    "N1.detectar_brecha: calcula Δ = NEED − HAVE (faltantes, incompatibilidades).",
    "N2.mapear_puentes: busca módulos que reducen la brecha (adaptadores, generadores).",
    "N3.elegir_minimo: compone el pipeline con el menor nº de saltos que satisface NEED.",
    "N4.validar: verifica invariantes (composición/adyacencia/orden, trazabilidad, coste).",
    "N5.ejecutar: corre pipeline en modo seleccionado (HAVE→NEED o NEED→HAVE).",
    "N6.calibrar: mide confianza/éxito y registra provenance; ofrece rollback."
  ],
  policy: {
    invariantes: ["composition","adjacency","ordering","evidence-only","provenance"],
    parcimonia: "min-saltos, min-adaptadores",
    confianza_umbral: 0.7,
    decision: "si confianza≥umbral → ejecutar; si no, sandbox/confirmación"
  }
}

nodes: [

  { id:"N0.modelar", rol:"Modelado de contratos",
    accion: [
      "Canonicalizar HAVE como {tipo, schema, capacidades, restricciones}.",
```

```
    "Canonicalizar NEED como {tipo, schema objetivo, invariantes, métricas de
aceptación}.",
    "Ej.: HAVE=dump JSONL; NEED=preprocessor-intake(schema_registro)."
   ],
   salida:["have*","need*"]
  },

  { id:"N1.detectar_brecha", rol:"Análisis de gap",
   accion: [
    "Comparar have* vs need* por campos, semántica y orden.",
    "Emitir faltantes, incompatibilidades y adaptaciones posibles."
   ],
   salida:["brecha{faltantes[],incompatibles[],costos}"], requiere:["N0.modelar"]
  },

  { id:"N2.mapear_puentes", rol:"Catálogo→candidatos",
   accion: [
    "Buscar transformadores/adaptadores (HAVE→NEED) y generadores (NEED→HAVE).",
    "Anotar pre/post y evidencia de uso."
   ],
   salida:["candidatos[]"], requiere:["N1.detectar_brecha"]
  },

  { id:"N3.elegir_minimo", rol:"Síntesis del pipeline",
   accion: [
    "Ordenar candidatos por parcimonia y cobertura de la brecha.",
    "Componer plan_conexion mínimo (ej.: pre-procesar→estructura →
constructor-de-estructuras).",
    "Marcar modo: 'adapt' (HAVE→NEED) o 'generate' (NEED→HAVE)."
   ],
   salida:["plan_conexion{steps[],modo}"], requiere:["N2.mapear_puentes"]
  },

  { id:"N4.validar", rol:"Garantías",
   accion: [
    "Simular salida y comprobar invariantes globales y del contrato NEED.",
    "Estimar confianza por evidencia/reglas/precedentes."
   ],
   salida:["validez{ok|fail,detalles}","confianza∈[0..1]"], requiere:["N3.elegir_minimo"]
  },

  { id:"N5.ejecutar", rol:"Aplicación del puente",
   accion: [
    "Si modo='adapt': transformar HAVE → artefacto que cumple NEED.",
    "Si modo='generate': materializar instancia desde NEED (plantillas/generadores)."
   ],
   salida:["artefacto_resultante","contratos_aplicados[]"], requiere:["N4.validar"]
  },
```

```
  { id:"N6.calibrar", rol:"Trazabilidad y ajuste",
    accion: [
      "Registrar provenance (fuentes, reglas, módulos).",
      "Ajustar confianza según resultado y guardar rollback."
    ],
    salida:["registro","confianza_final"], requiere:["N5.ejecutar"]
  }
]

modos: {
  "HAVE→NEED (adapt)": {
    ejemplo: "Tienes repo→ produces JSONL compatible (code-dump-exporter) para el
pre-procesador.",
    resultado: "HAVE normalizado al contrato requerido aguas abajo."
  },
  "NEED→HAVE (generate)": {
    ejemplo: "Tienes el contrato (schema_registro) → generas esqueleto de exporter en el
lenguaje elegido.",
    resultado: "Artefacto de implementación mínimo que satisface el contrato (scaffold +
pruebas de conformidad)."
  }
}

plantillas: [
  { nombre:"Puente mínimo válido", regla:"Prefiere 1 transformador directo a cadena de 2+;
documenta invariantes y evidencia." },
  { nombre:"Contrato primero", regla:"Siempre expresa HAVE y NEED como contratos
comparables antes de componer." },
  { nombre:"Rollback seguro", regla:"Cada paso deja snapshot para deshacer sin pérdida." }
]

métricas: {
  coste_puente: "nº pasos + latencia",
  cobertura_brecha: "faltantes_resueltos/total",
  confianza: "score por evidencia y precedentes",
  reusabilidad: "#veces que el puente se reutiliza en otros dominios"
}

interfaces: {
  provides: ["plan_conexion","artefacto_resultante","contratos","provenance","confianza"],
  requires: ["catalogo_modular","reglas_invariantes","politica_confianza"]
}

duo_logico (estructura complementaria): {
  arquetipos: [
    "Synth-Adapt (yin): partir de lo que hay para alcanzar el contrato",
    "Synth-Generate (yang): partir del contrato para materializar lo que falta"
```

```
  ],
  operador: "⊥-complement (bidireccionalidad): los dos modos comparten el mismo contrato
central y cambian la dirección de la flecha.",
  ecuacion: "Bridge = (HAVE → NEED) ∪ (NEED → HAVE) con invariantes compartidos",
  cierre: "Si el contrato evoluciona, ambos modos se actualizan en espejo."
}
```