Participant:10 - Interview Transcript

Behavior Trees (BTs)

Parallelism

> Concerning Behavior Trees, you disagreed with our statement that the parallel node in a BT corresponds to parallel execution, saying that "parallel in a BT is not the same as parallel execution in a program."
We specify that the parallelism implied in the model is logical parallelism and different from actual execution, since execution depends on the platform and hardware constraints. Would you agree?

It depends on whether you refer to logical parallelism or hardware-level parallelism. At the logical modeling level, what is possible is still constrained by hardware. If I define logic that cannot be executed by the hardware, then the model becomes meaningless in practice.

The key difference is that many BT implementations (for example, some ROS or Python-based libraries) do not support true parallelism due to language or runtime limitations. That influences even the logical interpretation of parallelism.

State machines and BPMN, in contrast, often provide formal support for logical parallelism. For BTs, there is not one single formal specification—parallel nodes may differ slightly depending on the implementation.

> So would you agree that clarifying the distinction between logical parallelism and execution-level parallelism would improve the precision of the paper?

Yes, adding such a clarification would help.

State Machines (SMs)

Skill Realization in SMs

> We clarify that:

- *Skill realization is implemented through the activity defined within a state.*
- *Communication refers to interactions between robots or external entities.*
- *Data refers both to internal data flow and data provided to states/actions.*

Would you agree with mapping a skill to the activity executed within a state?

Skills in state machines can be implemented as *do-actions*, or as entry/exit actions of states. Some state machine formalisms allow only entry and exit actions, others also allow “do” actions (executed while being in the state).

Many implementations do not exploit the full expressive power of the theoretical state machine formalism. The formalism itself is richer than many practical libraries.

> *So these capabilities belong to the formalism itself, not just implementations*

Mostly yes, they are part of the formalism, although tooling often does not support everything.

> *Would you agree that the expressive capabilities of state machines allow fine-grained control of robot behavior through entry, exit, and internal actions?*

Yes, state machines are extremely powerful in what they can express. The real question is whether this expressiveness becomes too complex in practice.

If you push too much complexity into a “skill,” you are not solving the complexity problem, you are just hiding it. The complexity is still there, only less explicit in the mission model.

Mission Concept Modeling in SMs (Emerging Question)

> *Does pushing preconditions and logic into skills reduce transparency at the mission level?*

Yes. If the notion of a skill is a capability, and you move all the logic inside the skill, then the mission model becomes simpler but hides complexity. That complexity becomes less visible and harder to analyze.

Guards and Choice Pseudostates

> *In state machines, if multiple guards on outgoing transitions from a choice pseudostate evaluate to true simultaneously, is there a default policy? Or is disambiguation up to the developer? Behavior Trees have intrinsic prioritization (left-to-right ticking). In SMs and BPMN, this prioritization is not intrinsic.*

Would you agree that: Guards should be mutually exclusive? A default transition should be explicitly modeled? Otherwise behavior is under-specified?

Yes, I agree. In state machines, guards are logical expressions, so you can explicitly define a default case (e.g., [not G1 and not G2]). But it is up to the developer to specify this.

Also, transitions usually require an event to trigger evaluation. Without an event, the guard will never be evaluated.

If writing this in the paper, make sure guards are clearly marked (e.g., with square brackets) and clarify the event triggering mechanism.

Time-Based Behavior in SMs

> Would you agree that time-based behavior (e.g., triggers every X seconds, delays) is only partially supported in classical state machines and often depends on implementations?

In UML state machines, time triggers are explicitly supported. A transition can be triggered after a certain time span or at a specific time. So time-triggered transitions are part of the formalism, not just implementation extensions.

> So would you classify this as full support?

It depends on what is supported in robotics practice. Formally, UML state machines do support time triggers.

Exception Handling and History States

> Can transitions from compound states serve as exception-handling mechanisms?

Yes. You can define transitions from higher-level (super) states to handle errors.

> What about task saving and resuming?

History states allow resuming execution at the last active substate of a compound state. You can: exit a superstate due to an error, execute recovery behavior, re-enter the superstate using a history state to resume where you left off.

This is supported by the formalism, though I am not sure which robotics implementations support it.

BPMN

Expressiveness

> You expressed neutral agreement regarding BPMN strengths and weaknesses. Should we add, remove, or rephrase some points?

I would object to claiming that BPMN has the richest expressiveness. State machines can express at least the same control-flow structures (parallelism, alternatives, conditions). Behavior Trees also allow conditional nodes.

I do not see constructs in BPMN that fundamentally cannot be modeled in state machines.

Frequent Condition Checking

> *We identified as a weakness that BPMN requires many boundary or intermediate events to model frequent reactive checks (e.g., battery level monitoring), making models complex. Would you agree?*

I would rephrase this weakness. It is not about reacting to highly frequent events, but about modeling parallel condition checks.

In BTs, continuous checking is intrinsic to the tick mechanism. In BPMN, to check something in parallel, you must explicitly model it, which can increase model complexity.

> *So the weakness is about explicit modeling of parallel monitoring conditions rather than frequency.*

Exactly.