Table of Contents

ReSpecT Overview ................................................................................................................................................................... 3
The ReSpecT Tuple Centre Coordination Model .................................................................................................................. 3
ReSpecT as a Core Coordination Language .......................................................................................................................... 3
ReSpecT Overview

ReSpecT (Reaction Specification Tuples) is a logic-based language for the coordination of complex software systems. ReSpecT promotes a coordination model providing tuple centres as programmable, general-purpose coordination media. The behaviour of ReSpecT tuple centres is programmed through the ReSpecT first-order logic language.

The ReSpecT Tuple Centre Coordination Model

A tuple centre is a tuple space enhanced with the possibility to program its behaviour in response to interactions. First of all, coordinated entities (ReSpecT agents, henceforth, or simply agents) can operate on a ReSpecT tuple centre in the same way as on a standard Linda tuple space: by exchanging tuples — which are ordered collection of knowledge chunks — through a simple set of coordination primitive. So, an agent can write a tuple in a tuple centre with an `out` primitive; or read a tuple from a tuple centre with primitives such as `in`, `rd`, `inp`, `rdp` specifying a tuple template - that is, an identifier for a set of tuples, according to some tuple matching mechanism. Reading tuples can be

- destructive — `in`, `inp` remove the matching tuple — or non-destructive — `rd`, `rdp` simply read the matching tuple
- suspensive — `in`, `rd` wait until a matching tuple is found — or non-suspensive — `inp`, `rdp` immediately return either the matching tuple or a failure result

but is anyway always non-deterministic: when more than one tuple in a tuple centre are found that match a tuple template, one is non-deterministically chosen among them and returned.

Accordingly, a tuple centre enjoys all the many features of a tuple space, which can be classified along three different dimensions:

- generative communication
- associative access
- suspensive semantics

The main features of generative communication (where information generated has an independent life with respect to the generator) are the forms of uncoupling (space, time, name) based on mediated interaction: sender and receiver do not need to know each other, to coexist in the same space or at the same time in order to communicate (to exchange a tuple, in particular), and more generally, to interact. Associative access (access based on structure and content of information exchanged, rather than on location, or on name) based on tuple matching promotes synchronisation based on tuple structure and content: thus, coordination is data-driven, and allows for knowledge-based coordination patterns. Finally, suspensive semantics promotes coordination patterns based on knowledge availability, and couples well with incomplete, partial knowledge.

Even more, while the basic tuple centre model is independent of the type of tuple, ReSpecT tuple centres adopt logic tuples — both tuples and tuple templates are essentially Prolog facts — and logic unification is used as the tuple-matching mechanism. So, for instance, an agent `ag1` performing operation `we ? in(activity(ag1,CaseID))` on tuple centre `we` containing tuples `activity(ag1,c16)` and `activity(ag2,c22)` will be returned tuple `activity(ag1,c16)` — the one unifying with the template — removed from `we`. Since the overall content of a tuple centre is a multiset of logic facts, it has a twofold interpretation as either a collection of messages, or a (logic) theory of communication among agents — thus promoting in principle forms of reasoning about communication.

Finally, a tuple centre is a programmable tuple space, so as to add programmability of the coordination medium as a new dimension of coordination. While the behaviour of a tuple space in response to interaction events is fixed — so, the effects of coordination primitives is fixed —, the behaviour of a tuple centre can be tailored to the system needs by defining a set of specification tuples, or reactions, which determine how a tuple centre should react to incoming / outgoing events. While the basic tuple centre model is not bound to any specific language to define reactions, ReSpecT tuple centres are obviously programmed through the ReSpecT logic-based specification language.

ReSpecT as a Core Coordination Language

The original ReSpecT is a logic-based language for the specification of the behaviour of tuple centre. As a behaviour specification language, ReSpecT:

- enables the definition of computations within a tuple centre, called reactions, and
ReSpecT - Overview

- makes it possible to associate reactions to events occurring in a tuple centre.

So, ReSpecT has both a declarative and a procedural part. As a specification language, it allows events to be declaratively associated to reactions by means of specific logic tuples, called specification tuples, whose form is \textit{reaction}(E,R). In short, given an event \textit{Ev}, a specification tuple \textit{reaction}(E,R) associates a reaction \(R \theta\) to \textit{Ev} if \(\theta = \text{mgu}(E, \text{Ev})\). As a reaction language, ReSpecT enables reactions to be procedurally defined in terms of sequences of logic reaction goals, each one either succeeding or failing. A reaction as a whole succeeds if all its reaction goals succeed, and fails otherwise. Each reaction is executed sequentially with a transactional semantics: so, a failed reaction has no effect on the state of a logic tuple centre.

All the reactions triggered by an event are executed before serving any other event: so, agents perceive the result of serving the event and executing all the associated reactions altogether as a single transition of the tuple centre state. As a result, the effect of a coordination primitive on a logic tuple centre can be made as complex as needed by the coordination requirements of a system. Generally speaking, since ReSpecT has been shown to be Turing-equivalent, any computable coordination law could be in principle encapsulated into a ReSpecT tuple centre. This is why ReSpecT can be assumed as a general-purpose core language for coordination: a language that could then be used to represent and enact policies and rules for coordination systems of any sort.

Adopting the declarative interpretation of logic tuples, a ReSpecT tuple centre has then a twofold nature a \textit{theory of communication} (the set of the ordinary tuples) and a \textit{theory of coordination} (the set of the specification tuples). In principle, this allows intelligent agents to reason about the state of collaboration activities, and to possibly affect their dynamics. Furthermore, the twofold interpretation (either declarative or procedural) of ReSpecT specification tuples allows knowledge and control to be represented uniformly (as Prolog-like facts) and encapsulated within the same coordination artefact.