What I want out of WebRTC NV

This is a strictly personal technical viewpoint. It has nothing to do with any of my roles in the standards process, except insofar those roles have helped me gather information.

Principles of the desired WebRTC NV

Positive principles:

- Easy things should be easy.
- Hard things should be possible.
- Interoperation with WebRTC-PC apps MUST be possible.
- Adding new transport protocols (non-SRTP, non-SCTP) should be possible.
- Adding new media types (depth), or new versions of old media types (SVC), should be possible.

Negative principles:

- No requirement to use SDP
- No requirement to use RTCTransceivers
- No requirement to use offer/answer

Unchanged principles:

- Exchange of setup information is out of scope for WebRTC. It's the app's problem.
- All information on the wire is encrypted, always. No built-in man-in-the-middle - even if attacker can eavesdrop on the setup information.
- Peer to peer communication should be the default. Communication via relays should be possible, to deal with NAT boxes and firewalls.

Practical implications of the principles above

In the current object model of WebRTC 1.0, the negative principles above lead to:

- Delete RTCPeerConnection
- Delete RTCTransceiver

This means that each of the magic powers that these objects have in terms of making changes to the state of other objects, or serving up summaries of the state of other objects, must now be made available through some interface - either on the existing objects, or on new objects.
The ultimate proof that we have arrived would be an RTCPeerConnection and an RTCTransceiver, both written in Javascript, that can be installed on top of the other objects

and permit apps written to run on WebRTC 1.0 to run without noticing a difference. But like the Universal Turing Machine, a practical implementation is not required.

Details

The RTCPeerConnection magic powers include:

- Triggering ICE candidate gathering, handling the resulting candidates gathered, and getting candidates to RTCTransports, via addCandidate, and handling the set of ICE servers used. This probably requires a new object that explicitly represents the ICE agent.
- Creating and configuring RTCIceTransport and RTCDtlsTransport objects.
- Collecting “unified states” over a set of RTC{Ice,DTLS}Transport objects. This has requirements on what info they expose about themselves.
- Creating and configuring RTCRTPSenders and RTCRTPReceivers via setLocalDescription/setRemoteDescription. This needs replacing via new init & update operations on the senders & receivers.
- Connecting Senders and Receivers to tracks and to transports
- As above, for datachannels
- Generating certificates - this is a static method, so can be attached anywhere convenient.
- Getting stats across all the objects involved, including for deleted objects.
  The latter part probably means some kind of “death gasp collection” interface to collect statistics from the objects when they are stopped/ended/removed; the former can be handled with an uniform “getStats()” function on each of the objects concerned. This may require exposing objects for stuff that has stats, but no JS objects.
- Identity provider operations.

The RTCTransceiver magic powers aren’t much. It has a SetCodecPreferences API, but that’s a helper that points into the SDP generation functions, which we already factor out - this doesn’t affect the RTPSender, which is controlled by RTCRtpParameters.codecs. So taking that away should be relatively easy.

Designing a proper API for the functions above shouldn’t be impossible. Not simple, but not impossible.

Making sure innovation can happen

When we want to make new protocols, codecs, transports and functions available, we will discover how our current objects, stats and interfaces are tied to the old models. Significant refactoring will have to occur - attempting to let the basic functions that are useful in all contexts have a common API, while the things that turned out to be protocol specific are broken out into subclasses. This will be an iterative process.

Putting it back together again

When we started the WebRTC project, the idealistic goal was a functionality that could grow for the future - that as new codecs, transports and functions become available, they are usable for applications immediately when both ends of the connection support them, without any need for application changes (as long as they don't want to take advantage of new functionality, of course).
We may see a need for maintaining something like WebRTC-PC and the SDP mappings of the functionality just for this purpose - making sure that "simple things remain simple" - even while the foundations are changing underneath it.

We'll see whether these goals are realistic or not as time goes by. Experience has shown us that our imagination frequently runs ahead of our resources.
With good work done, we will make progress.

.