Modes are relatively new elements of architectural description
  (introduced by D. Hirsch, J. Kramer, J. Magee and S. Uchitel)

Motivation:
  Modern systems are heterogeneous, widely distributed, highly dynamic
  Variety and dynamics of modern systems require flexible and adaptable architectural styles

Scenario-based abstraction
  A mode determines the runtime structural constraints
  e.g., interaction contexts for the completion of specific tasks
  A mode transition (from one mode to another) formalizes the evolution constraints
  e.g., admissible reconfigurations
This Talk

- We take the automotive case study in [EWSA’06]
  - Darwin notation
- We show a possible coding in ADR
  - ADR specs in disguise (in Darwin-like notation)
- We draw some conclusions
  - Modes vs ADR, Modes + ADR
Automotive Scenario: Route Planning Systems

On board navigation systems connected to a road-assistance service platform (in charge of providing route indications)

Components:

- **GPS**: Global Positioning System
- **HES**: Highway Emergency System
- **RPS**: Route Planning System
  it has three subcomponents:
  - **UI**: User Interface (displays info to the driver)
  - **UP**: User Prompt (receives from P and updates UI)
  - **P**: Planner (connects UP with the environment)
Automotive Scenario: Route Planning Systems

On board navigation systems connected to a road-assistance service platform (in charge of providing route indications)

Components:

- **GPS**: Global Positioning System
- **HES**: Highway Emergency System
- **RPS**: Route Planning System

It has three subcomponents:

- **UI**: User Interface (displays info to the driver)
- **UP**: User Prompt (receives from P and updates UI)
- **P**: Planner (connects UP with the environment)
Modes

- GPS, HES: none given
- RPS has three modes of operation
  - **autonomous**: connected to the GPS to establish the route (e.g. Road Sight scenario)
  - **convoy**: connected to another vehicle (e.g. Road Sight and Accident scenario)
  - **detour**: connected to the Highway Emergency system (HES) (e.g. Low Oil Level and Accident scenario)
- UI, UP: enable / disable
- P: master / slave
An Example Scenario

1NOTE: Darwin notation, Zoom required
Structural Constraints: Mode-Based Composition

RPS mode is related to the modes (and bindings) of its constituents

- Example:
  - RPS: autonomous
  - P: master
  - UP: enable
  - UI: enable

^2NOTE: Darwin notation
Autonomous RPSs are connected to the GPS
Detour RPSs are connected to the HES
Autonomous and detour RPSs are said leaders
Convoys are formed by a leader RPS followed by convoy RPSs

NOTE: Darwin notation, subcomponents not really needed
Possible RPS reconfigurations

- Detour ↔ Autonomous
- Convoy ↔ Autonomous
- Convoy → Detour

But RPS components cannot reconfigure independently:

- their constituents and contexts should reconfigure in such a way that the overall architectural constraints are respected
Possible RPS reconfigurations

- Detour ↔ Autonomous
- Convoy ↔ Autonomous
- Convoy → Detour

But RPS components cannot reconfigure independently:

- their constituents and contexts should reconfigure in such a way that the overall architectural constraints are respected
An Example of Reconfiguration: Detour2Autonomous

Read red as before and green as after, Zoom required
Basically, each pair (component type, mode) has a corresponding ADR type (and a suitable constructor)
(The cases of GPS and HES are similar)
An RPS in autonomous mode is a hierarchical component with a P in master mode, an UP in enable mode and an UI in enabled mode.

5Constructors for Convoy RPS and Detour RPS are similar
An RPS in convoy mode is a hierarchical component with a P in slave mode, an UP in enable mode and an UI in enabled mode.
An RPS in detour mode is a hierarchical component with a P in master mode, an UP in disable mode and an UI in enabled mode.
A System comprises:

- A GPS item
- An Autonomous Convoy Group (ACG) connected to the GPS
- A HES item
- A Detour Convoy Group (DCG) connected to the HES
An ACG can be

- either an RPS in autonomous mode \((\text{RPS-autonomous})\) followed by a convoy sequence \((\text{CS})\) of RPSs in convoy mode
- or two ACGs

\[^6\text{Constructors for DCGs and CSs are similar}\]
A DCG can be

- either a single RPS in detour mode followed by a sequence of RPSs in convoy mode
- or two DCGs
A CS can be
- either a single RPS in convoy mode
- or a concatenation of CSs
A single RPS in detour mode is *moved* from the group of RPSs in detour mode to the group of convoys if that the RPS reconfigures from mode detour to mode autonomous (d2a)

\[
x_5: \text{RPS-detour} \xrightarrow{\text{d2a}} y_5: \text{RPS-autonomous}
\]

\[
\text{system}(x_1: \text{GPS}, \; x_2: \text{HES}, \; x_3: \text{ACG}, \; \text{group}(x_4: \text{DCG}, \; \text{single}(x_5: \text{RPS-detour}, \; x_6: \text{CS})))
\rightarrow \text{system}(x_1: \text{GPS}, \; x_2: \text{HES}, \; \text{group}(x_3: \text{ACG}, \; \text{single}(y_5: \text{RPS-autonomous}, \; x_6: \text{CS})), \; x_4: \text{DCG})
\]

An RPS reconfigures from detour to autonomous mode if its UP reconfigures from disable to enable mode.

\[
x_2: \text{UP-disable} \xrightarrow{\text{d2e}} y_2: \text{UP-enable}
\]

\[
\text{rpsd}(x_1: \text{P-master}, \; x_2: \text{UP-disable}, \; x_3: \text{UI-enable})
\rightarrow \text{rpsa}(x_1: \text{P-master}, \; y_2: \text{UP-enable}, \; x_3: \text{UI-enable})
\]

An UI is ready to reconfigure from mode disable to mode enable.

\[
\text{uid} \xrightarrow{\text{d2e}} \text{uie}
\]

\footnote{Labelled reconfigurations can change the type!}
A single RPS in detour mode is *moved* from the group of RPSs in detour mode to the group of convoys if that the RPS reconfigures from mode detour to mode autonomous (d2a)

\[
\begin{align*}
x_5: \text{RPS-detour} & \rightarrow \neg \text{d2a} \rightarrow y_5: \text{RPS-autonomous} \\
\text{system}(x_1: \text{GPS}, x_2: \text{HES}, x_3: \text{ACG}, \text{group}(x_4: \text{DCG}, \text{single}(x_5: \text{RPS-detour}, x_6: \text{CS}))) & \rightarrow \text{system}(x_1: \text{GPS}, x_2: \text{HES}, \text{group}(x_3: \text{ACG}, \text{single}(y_5: \text{RPS-autonomous}, x_6: \text{CS})), x_4: \text{DCG})
\end{align*}
\]

An RPS reconfigures from detour to autonomous mode if its UP reconfigures from disable to enable mode.

\[
\begin{align*}
x_2: \text{UP-disable} & \rightarrow \neg \text{d2e} \rightarrow y_2: \text{UP-enable} \\
\text{rpsd}(x_1: \text{P-master}, x_2: \text{UP-disable}, x_3: \text{UI-enable}) & \rightarrow \neg \text{d2a}\rightarrow \text{rpsa}(x_1: \text{P-master}, y_2: \text{UP-enable}, x_3: \text{UI-enable})
\end{align*}
\]

An UI is ready to reconfigure from mode disable to mode enable.

\[
\begin{align*}
\text{uid} & \neg \text{d2e} \rightarrow \text{uie}
\end{align*}
\]

\(^7\text{Labelled reconfigurations can change the type!}\)
ADR Coding: Reconfiguration of an RPS

A single RPS in detour mode is *moved* from the group of RPSs in detour mode to the group of convoys if that the RPS reconfigures from mode detour to mode autonomous (d2a)

\[
x_5: \text{RPS-detour} \xrightarrow{-d2a} y_5: \text{RPS-autonomous}
\]
\[
\begin{align*}
\text{system}(x_1: \text{GPS}, x_2: \text{HES}, x_3: \text{ACG}, & \text{group}(x_4: \text{DCG}, \text{single}(x_5: \text{RPS-detour}, x_6: \text{CS}))) \\
\rightarrow \text{system}(x_1: \text{GPS}, x_2: \text{HES}, & \text{group}(x_3: \text{ACG}, \text{single}(y_5: \text{RPS-autonomous}, x_6: \text{CS})), x_4: \text{DCG})
\end{align*}
\]

An RPS reconfigures from detour to autonomous mode if its UP reconfigures from disable to enable mode.

\[
x_2: \text{UP-disable} \xrightarrow{-d2e} y_2: \text{UP-enable}
\]
\[
\begin{align*}
\text{rpsd}(x_1: \text{P-master}, x_2: \text{UP-disable}, x_3: \text{UI-enable}) \\
\rightarrow \text{rpsa}(x_1: \text{P-master}, y_2: \text{UP-enable}, x_3: \text{UI-enable})
\end{align*}
\]

An UI is ready to reconfigure from mode disable to mode enable.

\[
\text{uid} \xrightarrow{-d2e} \text{uie}
\]

7Labelled reconfigurations can change the type!
x2:UP-disable → y2:UP-enable

rpsd(x1:P-master, x2:UP-disable, x3:UI-enable)
-d2a→ rpsa(x1:P-master, y2:UP-enable, x3:UI-enable)

8 Read red as premise, green as conclusion
Conclusion: ADR vs Modes

- The main idea is to embed modes in ADR types.
- Additional types can account for common constructions (shapes, styles, patterns) such as sequences, sets, trees, etc.
- Such types can be used to define composition operations that determine the valid configurations.
- Types play a relevant role in reconfigurations (to preserve architectural constraints).

<table>
<thead>
<tr>
<th></th>
<th>Modes</th>
<th>ADR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Diagrams (UML, Darwin)</td>
<td>Graphs</td>
</tr>
<tr>
<td>Statics</td>
<td>Decl. Language (Alloy)</td>
<td>Algebra</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Policies (Ponder)</td>
<td>Rewrite rules</td>
</tr>
<tr>
<td>Consistency</td>
<td>Checked on run/design-time</td>
<td>By construction</td>
</tr>
</tbody>
</table>
Conclusion: ADR + Modes

- **Structured view:** ADR takes a global approach to glue all the architectural information in a suitable specification.

- **Projected view:** Modes follow a local approach that combines different views (diagrams, constraints, policies).

<table>
<thead>
<tr>
<th></th>
<th>ADR + Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Diagrams (UML, Darwin)</td>
</tr>
<tr>
<td>Statics</td>
<td>Algebra + Decl. Language</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Rewrite Rules + (Alloy), Policies (Ponder)</td>
</tr>
<tr>
<td>Consistency</td>
<td>Some by construction, some to be checked</td>
</tr>
</tbody>
</table>

Exploit structure and abstraction to rule over complexity:

- **Distill and organize in ADR** (interpreted over the desired language) as much architectural information (constraints, reconfigurations) as possible (in easy, non-cumbersome way).

- **Enrich the specification with the missing bits**
  - some architectural properties will be given by construction,
  - but others will still need analysis.
Conclusion: ADR + Modes

- **Structured view**: ADR takes a global approach to glue all the architectural information in a suitable specification.
- **Projected view**: Modes follow a local approach that combines different views (diagrams, constraints, policies).

<table>
<thead>
<tr>
<th></th>
<th>ADR + Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Diagrams (UML, Darwin)</td>
</tr>
<tr>
<td>Statics</td>
<td>Algebra + Decl. Language</td>
</tr>
<tr>
<td>Dynamics</td>
<td>Rewrite Rules + (Alloy), Policies (Ponder)</td>
</tr>
<tr>
<td>Consistency</td>
<td>Some by construction, some to be checked</td>
</tr>
</tbody>
</table>

Exploit structure and abstraction to rule over complexity:
- **Distill and organize in ADR** (interpreted over the desired language) as much architectural information (constraints, reconfigurations) as possible (in easy, non-cumbersome way).
- **Enrich the specification with the missing bits**
  - some architectural properties will be given by construction,
  - but others will still need analysis.
The end of the talk...

... is the beginning of the work (yet to be done!)

▶ comments, questions, requests for collaboration are all welcome

▶ thanks for the attention!