The New MATSim Routing Infrastructure

Thibaut Dubernet

Institute for Transport Planning and Systems (IVT)
ETH Zurich

MATSim User Meeting 2013
Why This Presentation?

- Recently, deep change in way to configure and use MATSim routing capabilities
- Lots of users got confused of not finding the old ways anymore
- The new infrastructure gives new possibilities, but most users not aware of it
  - formerly necessary hacks continue to be used
  - lots of code not compatible with complex trips, while making it so is now easy
- User Meeting seems the best place to diffuse this information
- Hopefully, this presentation can serve as a part of the documentation
Why a New Infrastructure?

- In activity-based mobility analysis, concept of trips and stages
  - trip: movement between two activities
  - stage: part of a trip performed with a unique mode
- MATSim plans: Activitys and Legs
  - Leg: primarily understood as trip, but no stage concept
- v0.2.0 (Fall 2010) “Detailed” Public Transport
  - actually simulate walk to stop, change from tram to bus...
  - stages represented by Legs, but MATSim understood Legs as trips
  - ⇒ very special case
    - special TimeAllocationMutator
    - special SubtourModeChoice
    - special ReRoute
    - ...

- ⇒ decision to make multi-stage trips a standard concept
- also the opportunity to clean old code
Nature of the Change

- idea: allow complex trips, without new data structures
- before: double calcRoute(Person p, Leg leg, Activity o, Activity d, double departure)
- now: List<PlanElement> calcRoute(Facility o, Facility d, double departure, Person p)
- Trips can contain activities: central way to identify those "dummy" activities
- "Main mode" vs "Leg mode": central way to identify main mode of a trip
- Trip: longest uninterrupted succession of Legs and "stage" Activities
Basic Components

- **TripRouter**
  - transmits routing requests to RoutingModules

- **TripRouterFactory**
  - configures a TripRouter by adding RoutingModules for all modes

- **RoutingModule**
  - computes trips for a given main mode

- **MainModeIdentifier**
  - gives the main/routing mode of a trip
  - attached to the TripRouter

- **StageActivityTypes**
  - identifies which activities are “stage activity” (e.g. “pt interaction”)

- related: **TripStructureUtils**
  - Provides trip-aware methods to get subtour structure, iterate through trips or (real) activities
Extension Concept

- Modification of the routing is done by replacing the TripRouterFactory
- This factory creates and configures the TripRouter to be used
- Designed to allow use of delegation
Some Possible Pitfalls

▶ When writing code which manipulates plans (e.g. replanning module), possibility:
  ▶ confusion Leg/Trip
  ▶ confusion leg mode/main mode
▶ do not forget to “declare” stage activities and main mode!
▶ no guarantee that one can get estimated travel times from the RoutingModules
▶ there may be some old replanning strategies not (yet) aware of those trips
Usecase: New “Teleportation” Mode

- we want to know what would be the impact of a public teleportation system on traffic.
  - one public teleportation station, immediate teleportation to the destination point
  - individuals take public transport to the station
- new multi-stage mode, with sub-trips using an existing mode
  - ⇒ demonstrates all the new features
  - ⇒ demonstrates the recommended patterns
- full code in tutorial.programming.example13MultiStageRouting
```java
public class SimulateTeleportation {
    public static void main(final String[] args) {
        final Controller controller = new Controller("path/to/my/config.xml");

        // create the teleportation station on a central link
        // on the PT tutorial scenario
        final Facility teleport =
            createFacility(
                new IdImpl("teleport"),
                controller.getScenario().getNetwork().getLinks().get(new IdImpl("2333")));

        // now, plug our stuff in
        controller.setTripRouterFactory(
            new MyTripRouterFactory(
                controller, teleport));
        controller.run();
    }
}
```
public class MyTripRouterFactory implements TripRouterFactory {
    public static final String TELEPORTATION_MAIN_MODE = "myTeleportationMainMode";
    private final Controller controller;
    private final Facility teleport;

    public MyTripRouterFactory(final Controller controller, final Facility teleport) {
        this.controller = controller;
        this.teleport = teleport;
    }

    @Override
    public TripRouter instantiateAndConfigureTripRouter() {
        final TripRouterFactory delegate = new TripRouterFactoryImpl(...);
        final TripRouter router = delegate.instantiateAndConfigureTripRouter();

        // add our module to the instance
        router.setRoutingModule(
            TELEPORTATION_MAIN_MODE,
            new MyRoutingModule(
                router,
                teleport));

        router.setMainModeIdentifier(
            new MyMainModeIdentifier(
                router.getMainModeIdentifier()));

        return router;
    }
}
Routing Module: Route Calculation

```java
@Deprecated
public List<PlanElement> calcRoute(
    final Facility fromFacility,
    final Facility toFacility,
    final double departureTime,
    final Person person) {
    final List<PlanElement> trip = new ArrayList<PlanElement>();

    // route the access trip
    trip.addAll(
        tripRouterDelegate.calcRoute(
            TransportMode.pt,
            fromFacility,
            station,
            departureTime,
            person));

    // create a dummy activity at the teleportation origin
    final Activity firstInteraction = createAct(STAGE, station, getLinkId());
    firstInteraction.setMaximumDuration(0);
    trip.add(firstInteraction);

    // create the teleportation leg
    final Leg teleportationLeg = createLegWithZeroDuration(
        TELEPORTATION_LEG_MODE,
        station.getLinkId(),
        toFacility.getLinkId());
    trip.add(teleportationLeg);

    return trip;
}
```
Routing Module: Stage Activities

```java
@Overrider
public StageActivityTypes getStageActivityTypes() {
    final CompositeStageActivityTypes stageTypes = new CompositeStageActivityTypes();

    // trips for this mode contain the ones we create, plus the ones of the
    // pt router we use.
    stageTypes.addActivityTypes(
        tripRouterDelegate.getRoutingModule(
            TransportMode.pt).getStageActivityTypes());
    stageTypes.addActivityTypes(new StageActivityTypesImpl(STAGE));

    return stageTypes;
}
```
Main Mode Identification

```java
public class MyMainModeIdentifier implements MainModeIdentifier {
    private final MainModeIdentifier defaultModeIdentifier;

    public MyMainModeIdentifier(final MainModeIdentifier defaultModeIdentifier) {
        this.defaultModeIdentifier = defaultModeIdentifier;
    }

    @Override
    public String identifyMainMode(final List<PlanElement> tripElements) {
        for (PlanElement pe : tripElements) {
            if (pe instanceof Leg && ((Leg) pe).getMode().equals(MyRoutingModule.TELEPORTATION_LEG_MODE)) {
                return MyTripRouterFactory.TELEPORTATION_MAIN_MODE;
            }
        }
        // if the trip doesn't contain a teleportation leg,
        // fall back to the default identification method.
        return defaultModeIdentifier.identifyMainMode(tripElements);
    }
}
```
Conclusion

- It is now possible to easily include new “complex” trips in MATSim
- This makes the detailed PT much easier to use
- Users (and developers) should be aware of it to get the full potential of MATSim
- One should not assume a strict Activity/Leg sequence anymore