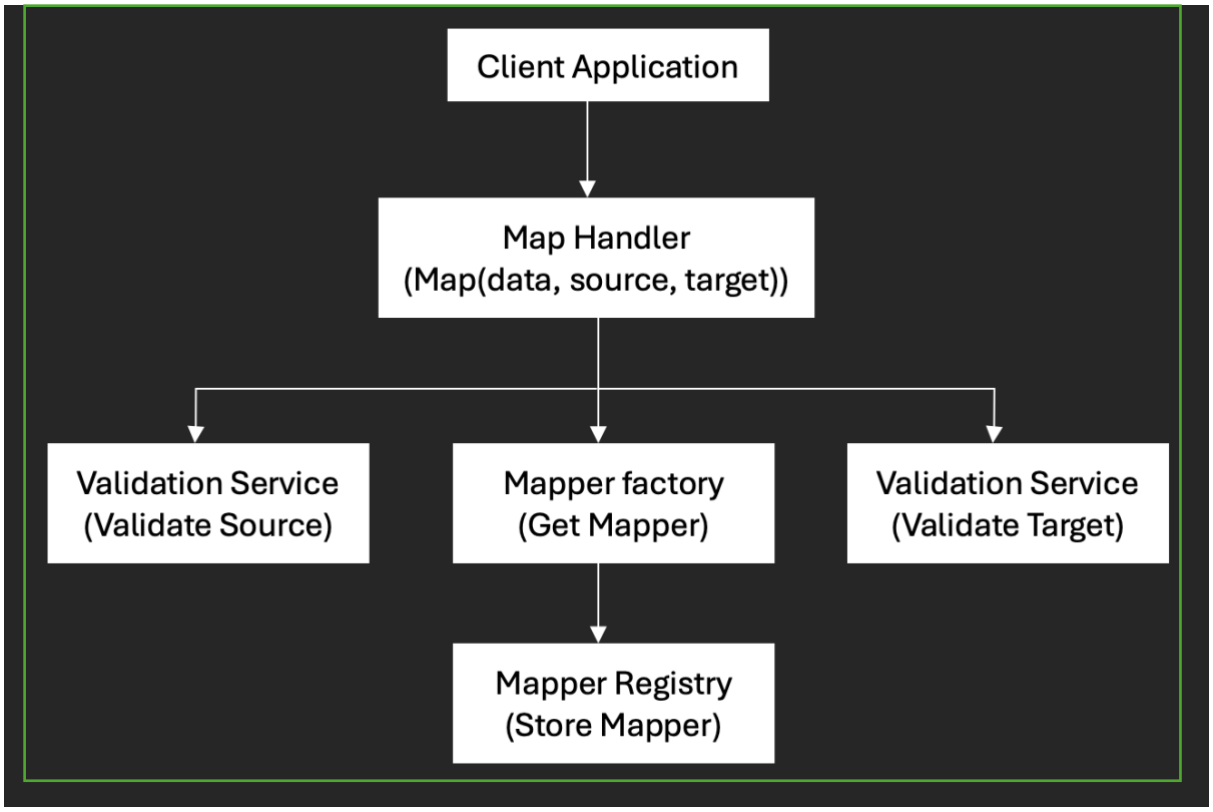
DIRS21 Mapping System Documentation

System Architecture

The mapping system uses a plugin-based architecture with these core components:

- **MapHandler**: Main entry point that orchestrates mapping operations
- **MapperRegistry**: Stores and retrieves mappers
- **ValidationService**: Handles pre and post-mapping validation
- **MapperFactory**: Creates mapper instances
- **Auto-discovery**: Automatically finds and registers mappers/validators at startup

Architecture Diagram



Auto-Discovery at Startup:

Key Classes and Methods

MapHandler

```
public object Map(object data, string sourceType, string targetType)
```

Orchestrates validation and transformation. This is the main method as per requirements.

IMapper Interface

```
string SourceType { get; }
string TargetType { get; }
object Map(object source);
```

Contract for all mapper implementations.

MapperBase<TSource, TTarget>

```
protected abstract TTarget MapInternal(TSource source);
```

Base class providing type-safe mapping.

IValidatorType

```
string TypeName { get; }
ValidationResult Validate(object data);
```

Self-registering validators.

ServiceCollectionExtensions

```
public static IServiceCollection AddDIRS21Mapping(this IServiceCollection
services)
{
    services.AddSingleton<IMapperRegistry, MapperRegistry>();
    services.AddSingleton<IValidationService, ValidationService>();
    services.AddSingleton<IMapperFactory, MapperFactory>();
    services.AddScoped<MapHandler>();
    return services;
}
// AddMappersFromAssembly - Scans assemblies and registers IMapper
implementations
// AddValidatorsFromAssembly - Scans assemblies and registers
IValidatorType implementations
```

```
public static IServiceCollection AddMappersFromAssembly(this
IServiceCollection services)
{
    // Scans all assemblies for IMapper implementations
    // Registers them as services.AddTransient(typeof(IMapper), mapperType)
    // Registers MapperInitializationService as hosted service
    services.AddHostedService<MapperInitializationService>();
    return services;
}

public static IServiceCollection AddValidatorsFromAssembly(this
IServiceCollection services)
{
    // Scans all assemblies for IValidatorType implementations
    // Registers them as services.AddTransient(typeof(IValidatorType),
validatorType)
    return services;
}
```

Usage in client App (comsole AppProgram.cs)

```
var host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((context, services) =>
    {
        services.AddDIRS21Mapping();           // ← Registers core services
        services.AddMappersFromAssembly();      // ← Auto-discovers mappers
        services.AddValidatorsFromAssembly();   // ← Auto-discovers
//validators
    })
    .Build();

await host.StartAsync(); // ← This triggers MapperInitializationService

var mapHandler = host.Services.GetRequiredService<MapHandler>(); // ← Get
from DI
```

MapperInitializationService

runs on startup:

- Gets all IMapper from DI
    - Registers with MapperRegistry
- Gets all IValidatorType from DI
    - Registers with ValidationService

Extending the System

To add a new partner (e.g., Booking.com):

Add in Client app or Library:

1.Create Model

```
public class BookingReservation
{
    public string BookingRef { get; set; }
    public string GuestFullName { get; set; }
    public string ArrivalDate { get; set; }
    public string DepartureDate { get; set; }
```

2. Create Mapper

```
public class ReservationToBookingMapper : IMapper
{
    public string SourceType => "Model.Reservation";
    public string TargetType => "Booking.Reservation";

    public object Map(object source)
    {
        var reservation = source as Reservation;
        return new BookingReservation
        {
            BookingRef = reservation.Id,
            GuestFullName = reservation.GuestName,
            ArrivalDate = reservation.CheckIn.ToString("yyyy-MM-dd"),
            DepartureDate = reservation.CheckOut.ToString("yyyy-MM-dd")
        };
    }
}
```

3. Create Validator (Optional)

```
public class BookingReservationValidator : IValidatorType
{
    public string TypeName => "Booking.Reservation";

    public ValidationResult Validate(object data)
    {
        // Validation logic
    }
}
```

That's it! The mapper and validator are automatically discovered - no registration needed.

Assumptions

1. **Type Naming**: All types follow pattern `{Domain}.{Entity}` (e.g., "Model.Reservation", "Google.Room")
2. **Models**: Models have public properties with getters and setters
3. **Mappers:** Stateless and thread-safe
4. **Auto-discovery**: All assemblies are scanned
5. **Validation**: Optional but recommended

Limitations

1. **Type Safety**: Map returns `object`, requires casting
2. **No Caching**: Mappers created on each request
3. **No Async Support**: Async method just wraps sync call
4. **No Configuration**: All mappings must be coded
5. **Validation Scope**: Object-level only, no field-level validation

Design Patterns Used

- **Strategy Pattern**: Each mapper is a strategy
- **Factory Pattern**: MapperFactory creates mappers
- **Registry Pattern**: Central mapper storage
- **Template Method**: MapperBase defines algorithm
- **Dependency Injection**: All dependencies injected

Repository

GitHub: https://github.com/narasegowdanithin/DIRS21.git

How to Run

```
# Clone the repository
git clone https://github.com/narasegowdanithin/DIRS21.git
cd DIRS21

# Build the solution
dotnet build

# Run the console application
dotnet run --project MappingConsoleUseCase
```