

AUTOSAR™

软件定义汽车 面向服务架构的应用迁移



龚小平

2023-03-16



BMW Group



BOSCH



DAIMLER



TOYOTA

VOLKSWAGEN
AKTIENGESELLSCHAFT

背景 - 软件定义汽车

汽车行业正在采用面向服务的架构（SOA）作为设计软件定义汽车（SDV）等现代应用的新范式

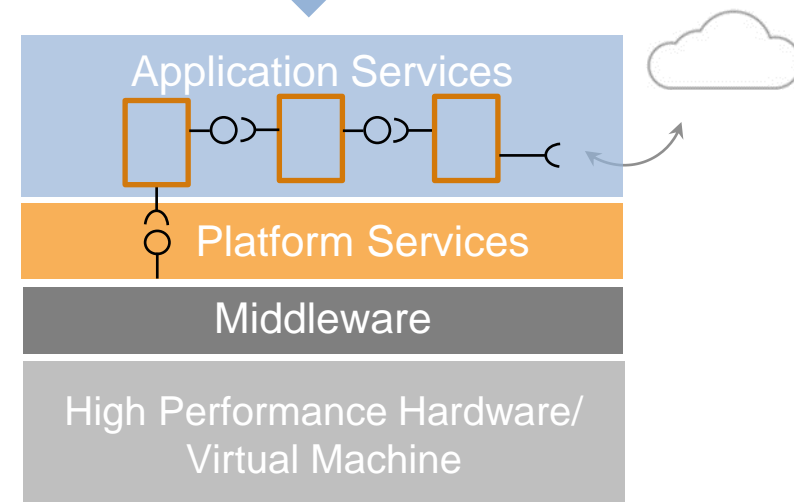


100110
001010
1001100010
001010
010010
100110
001010
010010



软件更新需求

- 频繁
- 可选
- OTA



更高的抽象:面向服务的架构

趋势 - 主机厂加大SOA自研力度

软件定义汽车



智能
网联
AI

基于中间件的SOA

C++

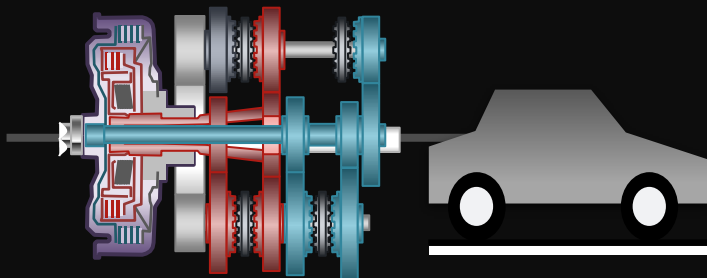


ROS

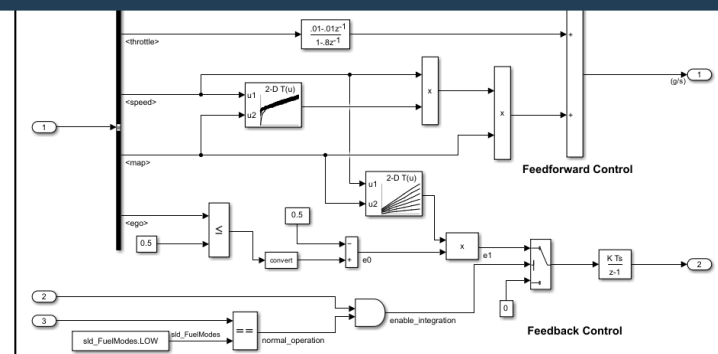


如何将传统的汽车软件应用迁移到基于 SOA 的服务?

基础车辆平台



控制
实时
CAN



内容

- 将传统汽车应用转换为 SOA 应用有哪些挑战?
- 如何将传统的应用软件组合分解为SDV的服务?
- 基于模型设计如何帮助将已有应用迁移到 AP服务?

内容

- 将传统汽车应用转换为 SOA 应用有哪些挑战?
- 如何将传统的应用软件组合分解为SDV的服务?
- 基于模型设计如何帮助将已有应用迁移到 AP服务?

为什么需要对已有应用实施迁移

重用经过验证的已有算法

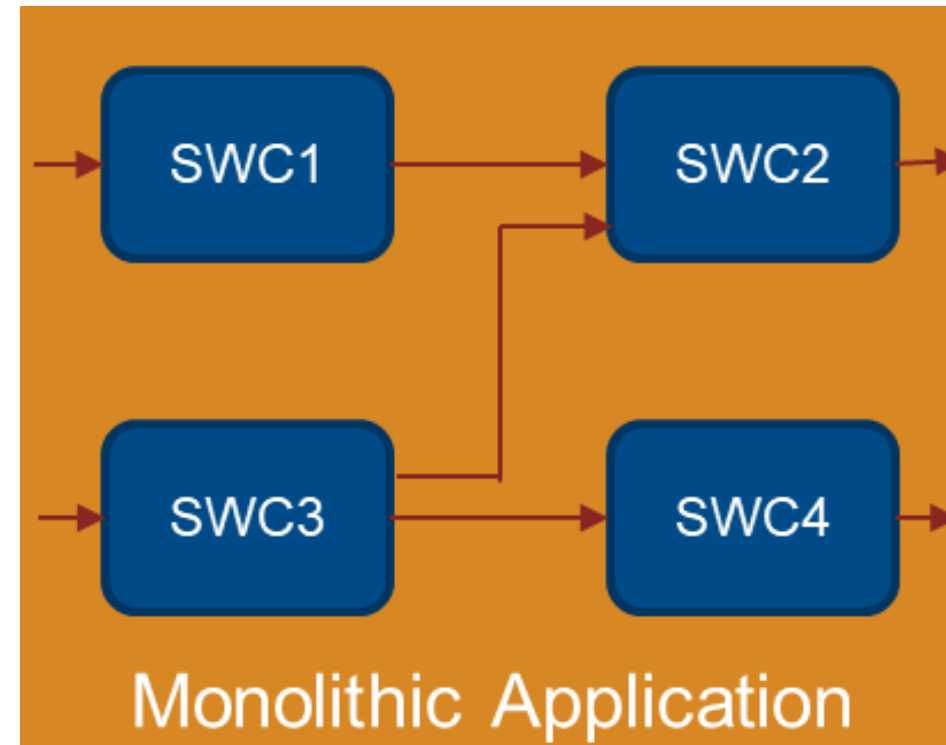
- 知识资产
- 算法库
- 作为扩展到高级功能的基线



已有应用本质上是单体式的

由各种软件组件组成

- 作为单个可执行文件部署在微控制器上
- 对不断变化的需求灵活性低



Reference: Software Architecture Patterns: Understanding Common Architecture Patterns and When to Use Them by Mark Richards

将传统汽车应用转换为SOA应用有哪些挑战

打破单体式架构

- 挑战 - 将旧应用程序的单体架构分解为更小、更模块化的服务。
- 要求 - 深入了解现有的应用程序架构以及服务化的设计原则。

识别和定义服务

- 挑战 - 确定应用需要提供的服务并清晰定义它们之间的接口。
- 要求 - 了解应用提供的功能以及定义服务化的工作流程。

确保兼容性

- 挑战 - 将旧应用程序转换为 SOA 应用程序可能会导致兼容性问题。
- 要求 - 确保新服务与现有服务兼容，并且应用程序仍可与生态系统中的其他系统通信。

管理数据

- 挑战 - 在旧应用程序中，数据可能与应用程序紧密耦合，因此难以作为单独的服务进行提取和管理。
- 要求 - 全面分析数据模型以及如何将其与应用程序逻辑分离。

性能和可扩展性

- 挑战 - 性能和可扩展性是迁移到 SOA 体系结构时的一个关键属性。
- 要求 - 确保新服务可以处理与现有应用程序相同的负载，并且可以根据需要进行扩展或缩减。

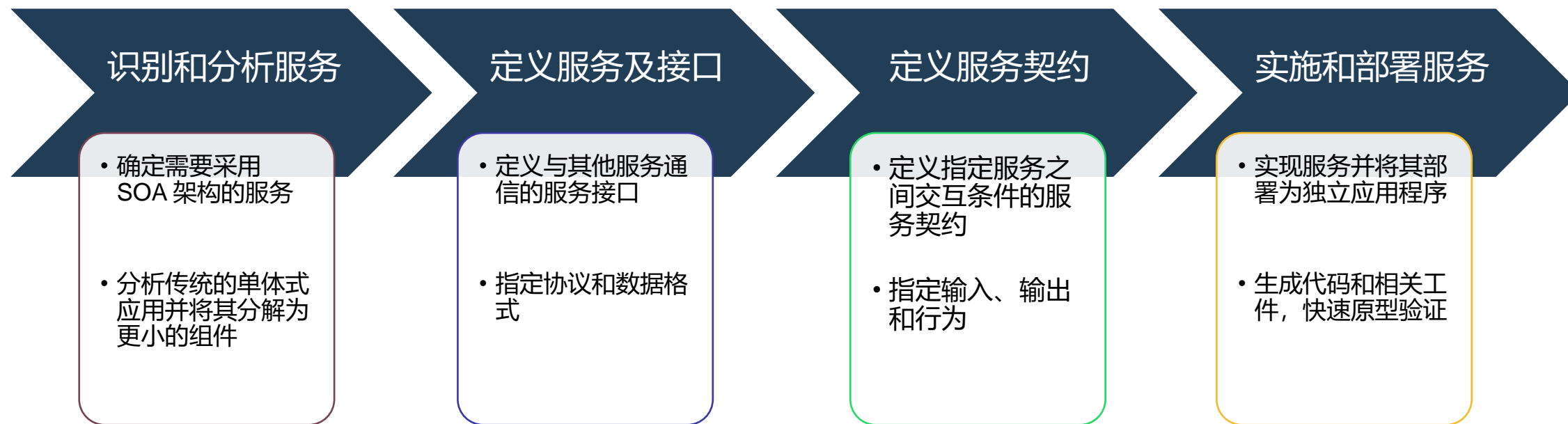
文化挑战

- 挑战 - 开发人员可能习惯于使用单体架构。
- 要求 - 需要学习新的架构和开发流程。

内容

- 将传统汽车应用转换为 SOA 应用有哪些挑战?
- 如何将传统的应用软件组合分解为SDV的服务?
- 基于模型设计如何帮助将已有应用迁移到 AP服务?

工作流程



设计原则 - SOLID

- **Single-Responsibility principle** (单一职责原则)
- Open-Close principle (开放闭合原则)
- Liskov substitution principle (里氏替换原则)
- Interface segregation principle (接口隔离原则)
- **Dependency inversion principle** (依赖倒置原则)

- *Explanation of Adaptive Platform Software Architecture R22-11*
- *Agile Software Development: Principles, Patterns, and Practices*

- 一个组件只负责功能的一个独立部分
- 减少接口变更需要, 提升可维护性

- 客户端不应该依赖于用不到的方法
- 将复杂接口按需拆分为简单接口

- 高层模块应该依赖于抽象而非低层模块
- 降低耦合, 提升复用
- 避免循环依赖

架构平台

组件层级

仿真

行为建模

函数定义

代码和架构文件生成

接口

The screenshot displays the AUTOSAR architecture platform interface. At the top, a toolbar includes various tool icons and a 'SIMULATE' section with 'Stop Time 10.0', 'Normal', 'Fast Restart', 'Run', and 'Stop' buttons. A red box highlights the 'Sensors' component in the 'tpc_composition' diagram, with a red arrow pointing to the '组件层级' (Component Level) label. A blue box highlights the 'Export' menu options, including 'Export Composition', 'Configure XML Options', 'Generate Code and ARXML', and 'Export ECU extract', with a blue arrow pointing to the '代码和架构文件生成' (Code and Architecture File Generation) label. A green box highlights the 'Schedule Editor' window, showing a sequence diagram for 'ThrottleControlComposition' with various function calls and timing constraints, with a green arrow pointing to the '函数定义' (Function Definition) label. A purple box highlights the 'Requirement Editor' window, showing a list of requirements, with a purple arrow pointing to the '接口' (Interface) label. The main workspace shows a block diagram with components like 'Sensors', 'Ctrl', and 'Actuator' connected by data flows. The AUTOSAR logo is visible at the bottom left.

内容

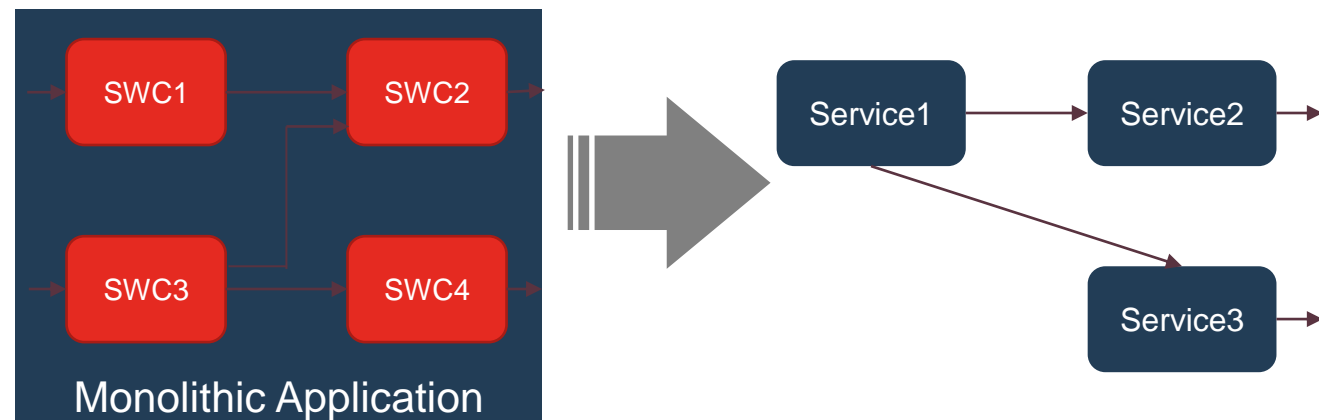
- 将传统汽车应用转换为 SOA 应用有哪些挑战?
- 如何将传统的应用软件组合分解为SDV的服务?
- 基于模型设计如何帮助将已有应用迁移到 AP服务?

识别和分析服务



要将单体式应用程序组件分解为服务，我们需要：

- 识别不同的组件、功能和依赖关系
- 了解组件交互和组件的执行顺序



Reference - <https://chrisrichardson.net/post/refactoring/2020/08/21/ten-principles-for-refactoring-to-microservices.html>

识别和分析服务

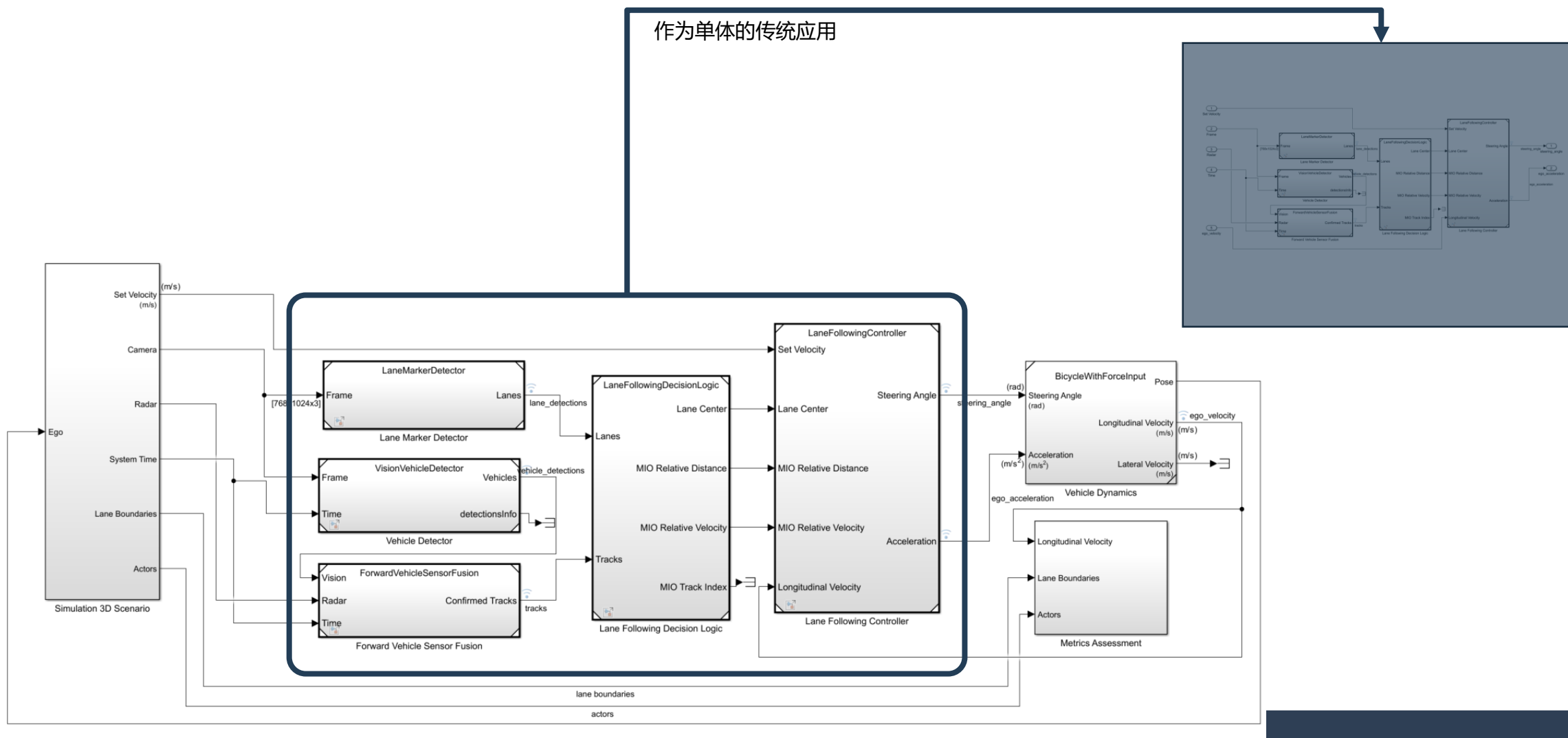
识别和分析服务

定义服务及接口

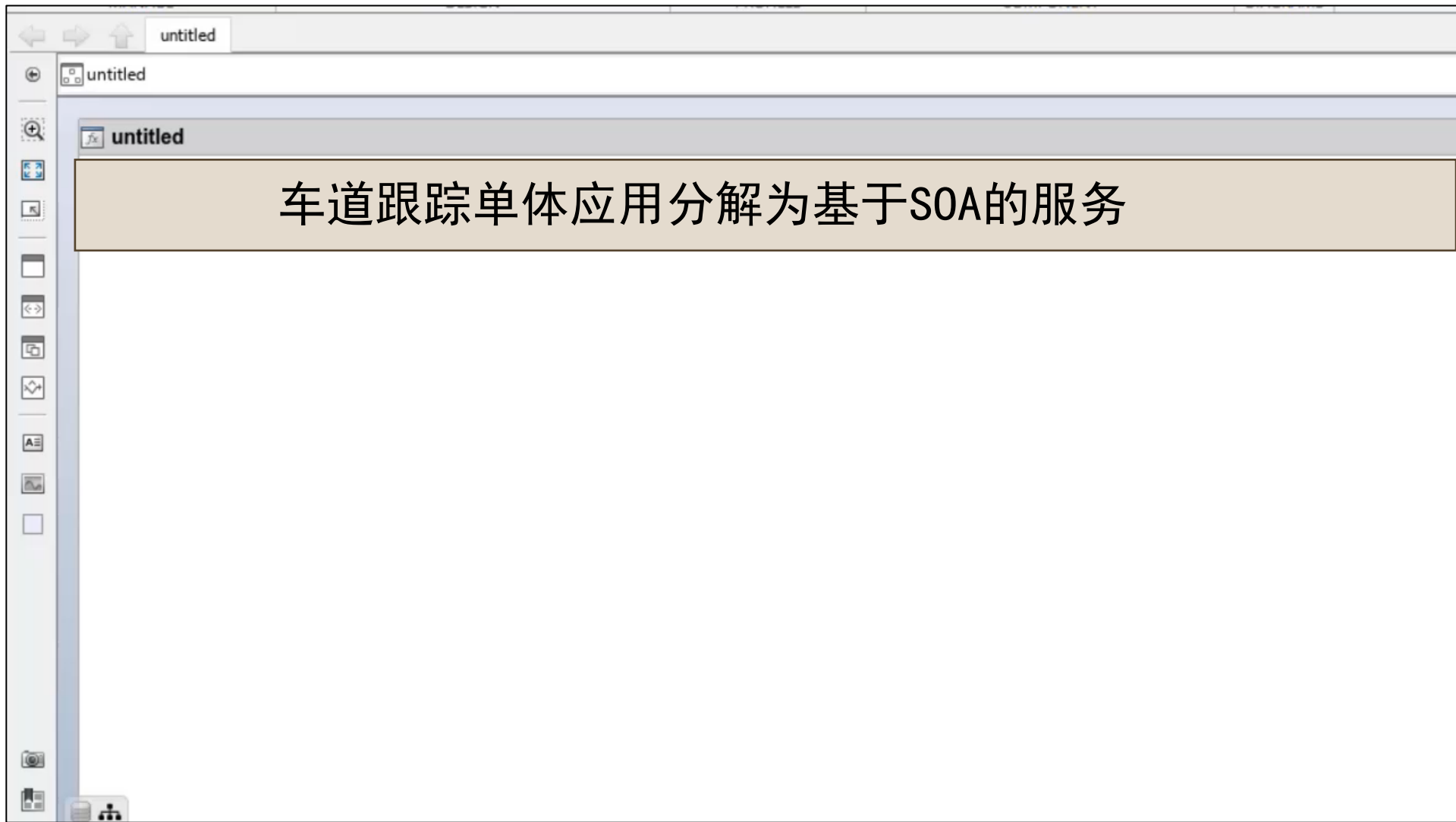
定义服务契约

实施和部署服务

作为单体的传统应用



识别和分析服务



定义服务接口



- 包含事件的数据元素和服务提供方提供的操作
- 作为服务的抽象与外部进行交互
- 具有功能性

	Type	Dimensions	Units	Complexity	Minimum	Maximum	
laneArchDD.sidd							
Adjustments							
status = Adjust(opts)							
status = Calibrate(opts)							
Calibrations							

配置服务接口

定义服务边界

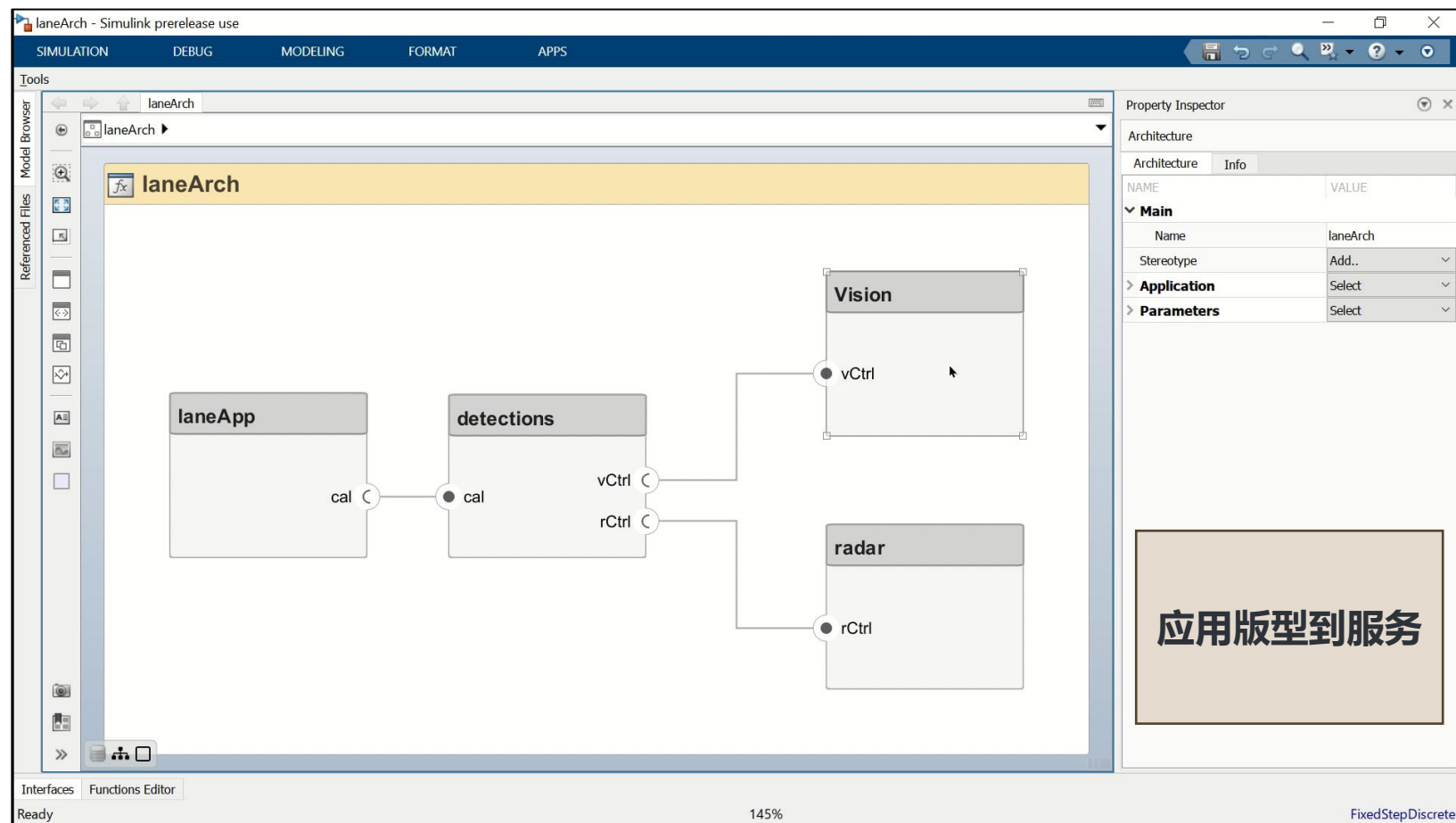
识别和分析服务

定义服务及接口

定义服务契约

实施和部署服务

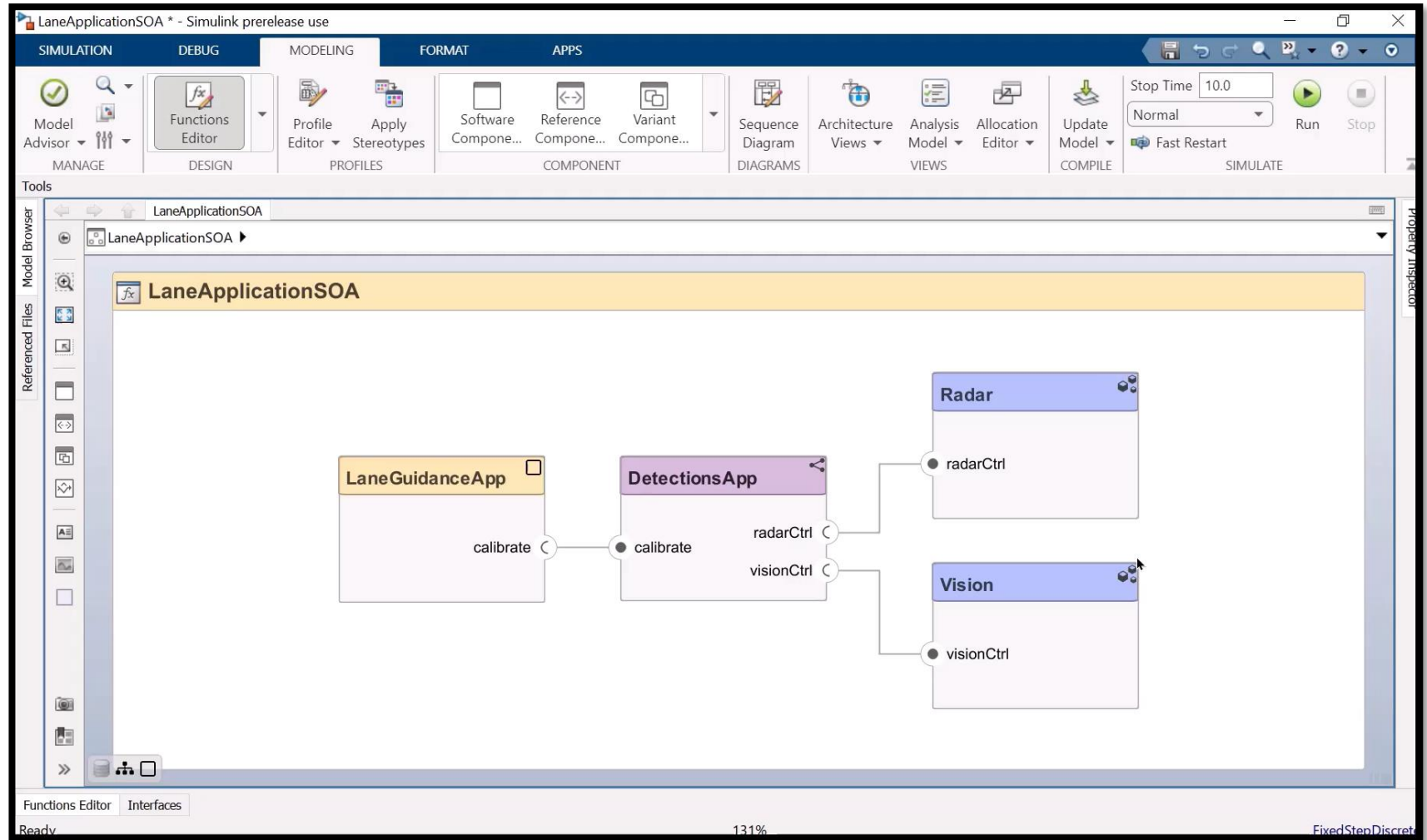
- 定义服务的职责、范围和依赖关系等
- 作为服务接口的补充
- 包括非功能属性



定义服务契约



- 指定服务的输入、输出和行为
- 可以看作服务原型
- 服务双方并行的前提



实现和部署服务

识别和分析服务

定义服务及接口

定义服务契约

实施和部署服务

- 配置中间件 (AP) 属性

- 检查映射关系

- 生成C++代码

The screenshot displays the Simulink environment for a project named 'LaneApplicationSOA'. The main workspace shows a block diagram with the following components and connections:

- LaneGuidanceApp** (yellow box) containing a sub-block **LaneGuidanceApp** with a 'calibrate' output.
- DetectionsApp** (purple box) containing a sub-block **DetectionsApp** with a 'calibrate' input and two outputs: 'radarCtrl' and 'visionCtrl'.
- Radar** (blue box) containing a sub-block **Radar** with a 'radarCtrl' input.
- Vision** (blue box) containing a sub-block **Vision** with a 'visionCtrl' input.

Connections are shown as lines with circular endpoints: 'calibrate' from LaneGuidanceApp to DetectionsApp; 'radarCtrl' from DetectionsApp to Radar; and 'visionCtrl' from DetectionsApp to Vision.

The **Property Inspector** on the right shows the configuration for the **Main** architecture:

NAME	VALUE
Main	
Name	LaneApplicationSOA
Exported Composition Name	LaneApplicationSOA
Stereotype	Add..
Parameters	
	Select

The status bar at the bottom indicates 'Ready', '77%' zoom, and 'FixedStepDiscrete' mode.

实现和部署服务

每个服务都需要部署为一个独立的应用程序并生成相关工件：

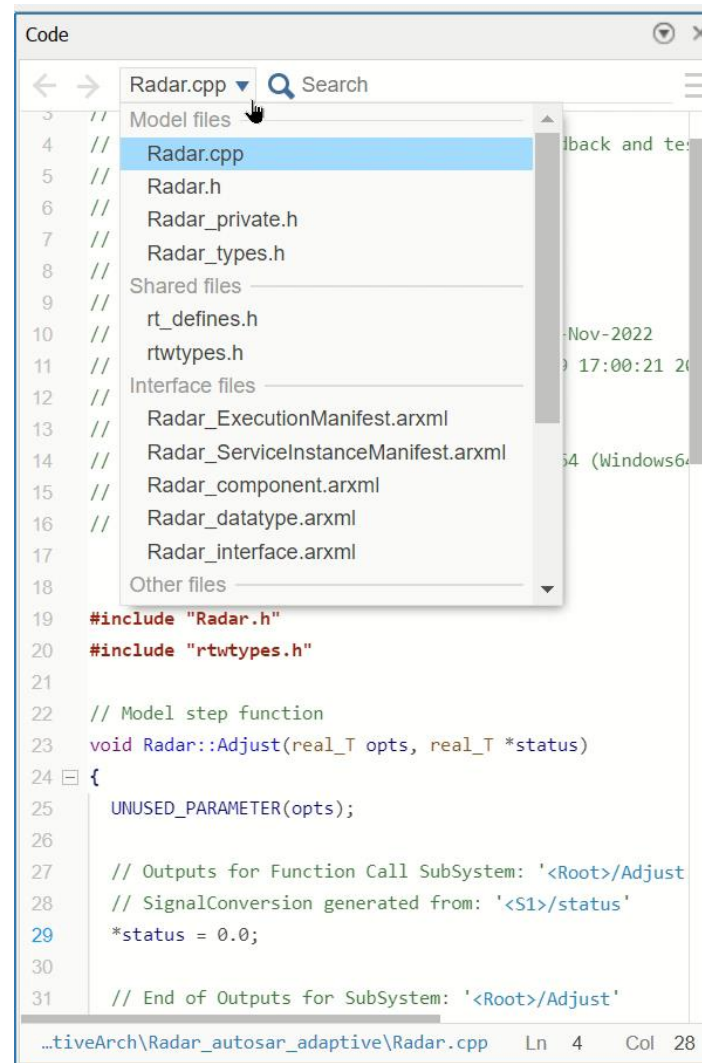
- 接口描述
 - 接口元素和数据类型
- 机器清单
 - 服务的运行环境
- 执行清单
 - 服务如何运行
- 服务实例清单
 - 服务如何通信

识别和分析服务

定义服务及接口

定义服务契约

实施和部署服务



The screenshot shows a code editor window titled 'Code' with a search bar and a dropdown menu. The dropdown menu is open, showing a list of files including 'Radar.cpp', 'Radar.h', 'Radar_private.h', 'Radar_types.h', 'Shared files', 'rt_defines.h', 'rtwtypes.h', 'Interface files', 'Radar_ExecutionManifest.arxml', 'Radar_ServiceInstanceManifest.arxml', 'Radar_component.arxml', 'Radar_datatype.arxml', 'Radar_interface.arxml', and 'Other files'. The 'Radar.cpp' file is selected in the dropdown. Below the dropdown, the code for 'Radar.cpp' is visible, starting with '#include "Radar.h"' and '#include "rtwtypes.h"', followed by a function definition for 'Adjust'.

```
Code
< > Radar.cpp Search
3 // Model files
4 // Radar.cpp
5 // Radar.h
6 // Radar_private.h
7 // Radar_types.h
8 // Shared files
9 // rt_defines.h
10 // rtwtypes.h
11 // Interface files
12 // Radar_ExecutionManifest.arxml
13 // Radar_ServiceInstanceManifest.arxml
14 // Radar_component.arxml
15 // Radar_datatype.arxml
16 // Radar_interface.arxml
17 // Other files
18
19 #include "Radar.h"
20 #include "rtwtypes.h"
21
22 // Model step function
23 void Radar::Adjust(real_T opts, real_T *status)
24 {
25     UNUSED_PARAMETER(opts);
26
27     // Outputs for Function Call SubSystem: '<Root>/Adjust'
28     // SignalConversion generated from: '<S1>/status'
29     *status = 0.0;
30
31     // End of Outputs for SubSystem: '<Root>/Adjust'
```

实现和部署服务

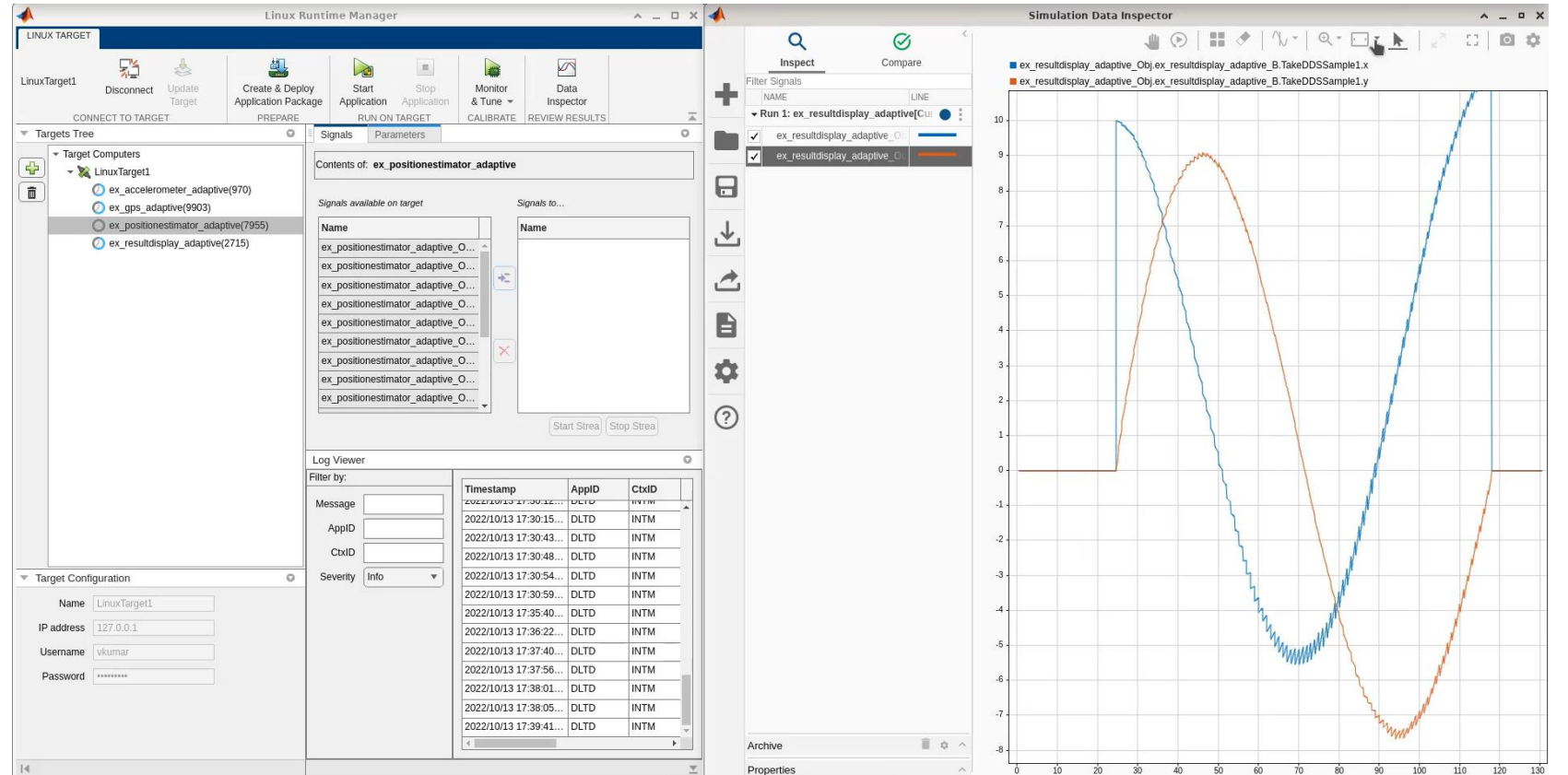
识别和分析服务

定义服务及接口

定义服务契约

实施和部署服务

- 快速原型部署
- 服务运行控制
- 测量和标定



总结

- 将传统汽车应用转换为 SOA 应用所面临着诸多挑战
- 需要结合设计原则和工程经验开展服务的识别和分析
- 选择合适的流程、方法和工具平台实施迁移工作
- 基于模型设计提供了SOA迁移的关键能力
 - 服务的定义和建模
 - 自动生成C++代码
 - 快速原型部署和验证

MATLAB EXPO

中国

5月30日 | 上海

6月06日 | 北京

Register at matlabexpo.cn



© 2023 The MathWorks, Inc.



获取更多MATLAB & Simulink在AUTOSAR的应用