Experiences with Automotive Service Modeling

Akihito Iwai
DENSO CORPORATION
1-1, Showa-cho, Kariya-shi,
Aichi-ken, 448-8661, Japan
+81 566 61 5312
akihito_iwai@denso.co.jp

Norio Oohashi
NEC Corporation
7-1, Shiba 5-chome, Minato-ku,
Tokyo 108-8001, Japan
+81 3 3454 1111
noh@ab.jp.nec.com

Steven Kelly
MetaCase
Ylistönmäentie 31
FI-40500 Jyväskylä, Finland
+358 14 641 000
stevek@metacase.com

ABSTRACT
Existing component-based development in the automotive world is showing the strain, as systems grow ever larger and start to interact with systems in the world outside the vehicle. A service-oriented approach offers benefits of modularity and runtime configurability, but raises challenges of a suitable language and platform. We examine the applicability of BPEL to automotive services. From our preliminary results we suggest the need for Domain-Specific Modeling to better address the particular requirements of the automotive service domain.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**] Design Tools and Techniques - *user interfaces, state diagrams* D.2.6 [**Software Engineering**] Programming Environments - *programmer workbench, graphical environments* D.3.2 [**Programming Languages**] Language Classifications - *Specialized application languages, very high-level languages*

General Terms
Languages, Experimentation.

Keywords
Domain-specific modeling, language design, service orientation

1. INTRODUCTION
This paper describes research in progress by DENSO, NEC and MetaCase on modeling languages to improve the development of automotive software. In particular, we are focusing on the creation and integration of services that communicate with vehicles, yet exist at least in part outside them.

This kind of communication becomes more important year by year, with the rise of large-scale automotive systems such as Telematics, Vehicle-to-Vehicle, Vehicle-to-Infrastructure and similar systems with outside cooperation. Such systems are *not* deterministic: they have dynamic behavior, as each component is expected to adapt to its environment, leading to the whole system being in a constant state of flux.

The specific modeling and verification needs for these systems are not met well by existing standards and development approaches within the automotive industry. We thus looked outside our industry, and our research suggested a service-oriented approach to design and verification.

In particular, we decided to try specifying services with Business Process Execution Language (BPEL) [7], a standard from the IT industry for modeling the interaction of services. Another factor motivating the use of a standard from the IT industry is that it may enable us to follow the rapid evolution of service integration as soon as possible.

This paper describes the results to date of an ongoing research project called 'DARWIN' which is based on a SOA (Service-oriented Architecture) approach [1][3][4][6]. In the project we have tested some of the capabilities of BPEL and attempted to validate its applicability to our domain based on some automotive use cases.

2. DOMAIN
In this section we explain why services are becoming important in automotive development and what are the particular challenges relating to their adoption.

2.1 Need for a Service Integration Platform
The current approach, component-based development, has served automotive developers well, allowing reuse, the division of labor, and insulation from the internal details of implementations. However, as the amount of functionality in and around vehicles has grown, component-based development has begun to be strained. Every time a new service or technology is developed, a software module to connect each service must be added, and gradually the system becomes increasingly complex, making it difficult to develop and integrate new services.

In particular, component-based development assumes that at design time there is already complete knowledge of all the components that the system will contain at runtime. It is thus not well able to cope with the introduction of new functionality over the lifetime of a vehicle, or of functionality provided by third parties.

To ameliorate these problems, automotive engineers are looking at a service-oriented approach. A layer is introduced on which workflows (called service processes) integrate services and technologies. The layer makes service processes easy to add or change, and consequently it can be expected to make possible the emergent evolution of new services.

2.2 Challenges for Service Integration
One of the key challenges for Service Integrated Systems is the modeling of the services and their interaction. In the automotive domain, they must also balance the strong yet conflicting requirements of security and dynamic, autonomous configuration. These challenges are listed in Table 1.

Table 1. Challenges for Service Integration

- Service Modeling
  - Service model definition and implementation
  - Abstract model of vehicle service
  - Capturing requirements from multiple stakeholders
  - Developed by multiple vendors
- Secure Platform
  - Protection mechanism against invalid external access
  - Highly dependable OS
  - Firewall
- Pervasive Computing
  - Adapting to dynamic change of system configuration
  - Installing ad-hoc communication system
  - Dynamic configuration

2.3 Intelligent parking service as use case

In the rest of this article we will take an Intelligent Parking Service as our case study. In this service and its related environment, the car, service provider and mobile phone work collaboratively to provide parking navigation, remote security and road pricing. The use case can be walked through by following the car from right to left in Figure 1. When the car approaches the parking lot, the Parking Navigation Service guides it to a free space. While parked, the Remote Security Service is running and can be accessed via a Smart key and monitor. When exiting, the user accesses the Road Pricing Service. At each stage, the car autonomously detects, provides, and integrates with appropriate services according to the requirements of the situation.

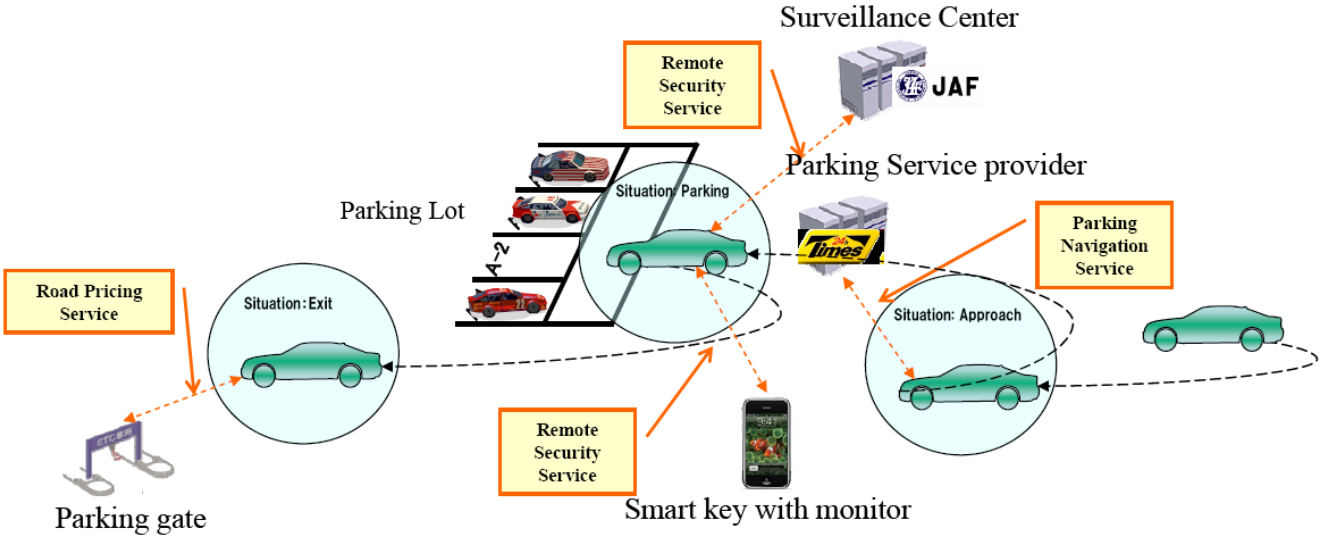

Figure 1. Intelligent Parking Service and related environment

3. EXPERIENCES WITH BPEL

BPEL, or more properly WS-BPEL, is a standard for describing business process orchestration. The standard itself uses an XML representation, but tools exist allowing a graphical representation. The similar BPMN has a graphical notation, and at least for simple cases can be used interchangeably with BPEL; with more complex graphs, some structures become difficult or impossible to represent in human-readable BPEL.

3.1 Problems with BPEL

Our experiences with BPEL indicate that it is insufficient for the requirements of automotive services. Below we list and explain some of the key areas in which we have found that BPEL falls short of our needs. These are not intended as criticisms of BPEL when applied to the domains it was intended for, only of problems when applying it to our domain.

- BPEL has no facilities for describing the dependability of a service, such as real-time guarantee, safety, reliability, and security. Automotive modeling strongly requires this capability.
- BPEL has no model of resources. Many of the choices we want to make in the models are based on whether resources are available, and their properties: e.g. network bandwidth and latency, 3D graphical display in the car, or text-to-speech function in the car.
- BPEL has no native facilities for autonomous choice among multiple possible services. Start and end conditions had to be expressed outside of BPEL.
- BPEL has no facilities for hot swapping from one implementation of a service to another, copying the current state into the new service implementation.
- BPEL has poor facilities for fault tolerance, e.g. modeling the behavior of a system with failures. Higher-level facilities than try-catch would be needed.
- BPEL has poor facilities for splitting a model into multiple parts, with each part only ultimately being decided at runtime. The underlying assumption in BPEL is more that the whole model of a service is available in one place at design time.

Of course, we are not saying that modeling our systems in BPEL would be impossible. Systems can be built with far fewer and less domain-specific concepts, right down to binary ones and zeros. The challenge is more to find a language that is at the highest possible level of abstraction that still allows sufficient precision and freedom.

Our attempts to use BPEL to model our applications left us with the impression that this is not the best place to apply BPEL, nor is BPEL the best language to apply to this problem. Neither result is particularly surprising, but it seemed wise to try a "standard" first, rather than risk reinventing the wheel.

3.2 Attempt to minimally extend BPEL

Our first attempt to resolve the issues we found was to make the minimum possible extensions to BPEL to add DARWIN service concepts. Figure 2 shows an example diagram; the corresponding BPEL XML translation is shown in Listing 1.

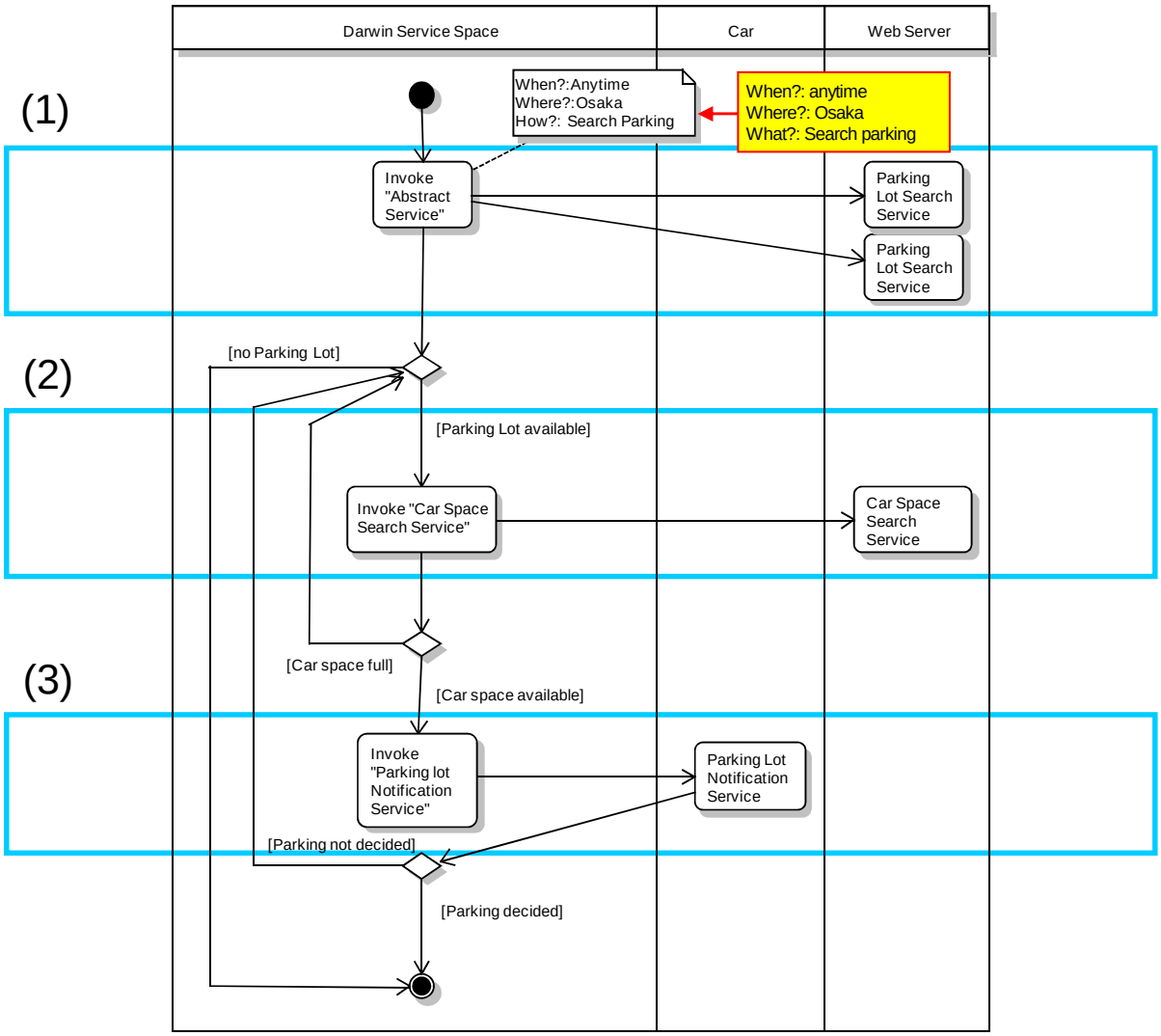

Figure 2. Flowchart diagram of parking navigation service

To keep close to the standard, we tried to make the minimum possible changes to the modeling language itself, e.g. by using (abusing!) comment objects to contain extra information that was needed for our domain. An example can be seen at the top, to the right of the Start symbol: the context information for the service is provided in a particular text syntax in the comment (the box containing the translation into English was added later for this article). The corresponding additions to the BPEL XML schema can be seen as attributes in the first <invoke> tag in Listing 1. The numbers in parentheses in the listing indicate the corresponding grouped sections from the diagram.

Listing 1. BPEL XML of parking navigation service

(1)

```
<invokeAbstractService when="always" where="area:Osaka"
          what="Search parking" execute="all" timing="start">
  <params>
    <param type="int">latitude</param>
    <param type="int">longitude</param>
  </params>
  <return type="string">ParkingServiceName</return>
</invokeAbstractService>
```

(2)

```
<invoke name="InvokeNotifyEmptySpaceNumber"
          partnerLink="ParkingServer"
          operation="GetEmptySpaceNumber"
          portType="GetEmptySpaceNumberPT"
          inputVariable="ParkingServiceName"
          outputVariable="ParkingNumber">
</invoke>
```

(3)

```
<invoke name="InvokeCheckParkingCar" partnerLink="CAR"
          operation="CheckParkingCar"
          portType="CheckParkingCarPT"
          inputVariable="ParkingNumber"
          outputVariable="bParkCar">
  <toParts>
    <toPart part="partnerLinkName"
          toVariable="ParkingServiceName" />
    <toPart part="partnerLinkName" toVariable="ParkingNumber" />
  </toParts>
</invoke>
```

We found the results to be unsatisfactory: even this sample is known to be incorrect in parts. To be executable, the BPEL needs exact values, but in a freeform comment field it is all too easy to enter something that is close but not correct. For instance, the "when" attribute should have the value "always", as in the XML; in the model, this has mistakenly been entered as "anytime". Trying to make a generator cope with all possible Japanese translations of "always" would be a losing battle.

It became clear to us that it is not possible to change a standard modeling language to be more appropriate to a given domain, whilst still keeping it as the standard: you can't have your cake and eat it!

4. PROPOSED SOLUTIONS

Although we have initially used BPEL because of its existing position as a standard, we find that it is not sufficiently well-suited to our needs. This is no particular criticism of BPEL: although in theory it was designed to be applicable to all kinds of service and workflow modeling, in practice its creators had in mind the IT domain. It is only natural that outside that domain it will not perform as well.

In some aspects BPEL is thus too generic, lacking concepts that we need; in other aspects it contains concepts that are specific to the IT domain. Those concepts are either unnecessary for us, or (worse) twist the meaning of a concept common to both domains in a way that is unsatisfactory in our domain.

We believe that Domain-Specific Modeling [9] will be needed for satisfactory modeling of automotive services. The details of a satisfactory language are still to be finalized, but there seem to be two main approaches to its construction. We can either fundamentally extend BPEL – adding new concepts, removing unnecessary ones, and customizing existing ones – or we can make a new DSM language from scratch. While there will of course be some similarities with BPEL in certain areas, since we are still in the service and process modeling domain, having our own DSM language would give us freedom to express things in the way that best fits our needs.

Our previous experience with DSM using MetaEdit+ [10] indicates that creating such a language and tool support is no slower, probably even faster, than our attempt to twist the existing BPEL language and tool to fit our purposes.

4.1 Modeling language idea

BPEL is just one of many languages, and for us language is just a way to solve the requirements. In other words, the requirements are the most important thing. The definition of all the required information about the service in a model is thus the starting point for our language. To support the runtime discovery and orchestration of autonomous services, the language should offer good concepts for modularity.

We will not be copying an existing language, but we will certainly use our knowledge of BPEL and other languages such as SRML [2] to inform our design process. We feel this is the logical extension of the advice in [8] to reuse or extend existing languages where possible: where that is not possible, create a new language, but do not throw away lessons learned from earlier languages.

4.2 Example of service description

Figure 3 shows how an example parking navigation service integrates with the wider platform. The Service Process Manager (SPM) is composed of the Darwin Service Space (DSS) and Situation Inference Engine (SIE). Within the DSS there are many possible autonomous services, discovered at runtime and assessed for suitability by the SIE.

At the highest level of abstraction, the model of a service can be similar to BPEL – or indeed any other process modeling language. Even here, however, there are elements that do not sit well with BPEL, such as the Start Condition used by the SIE; we will discuss this next.

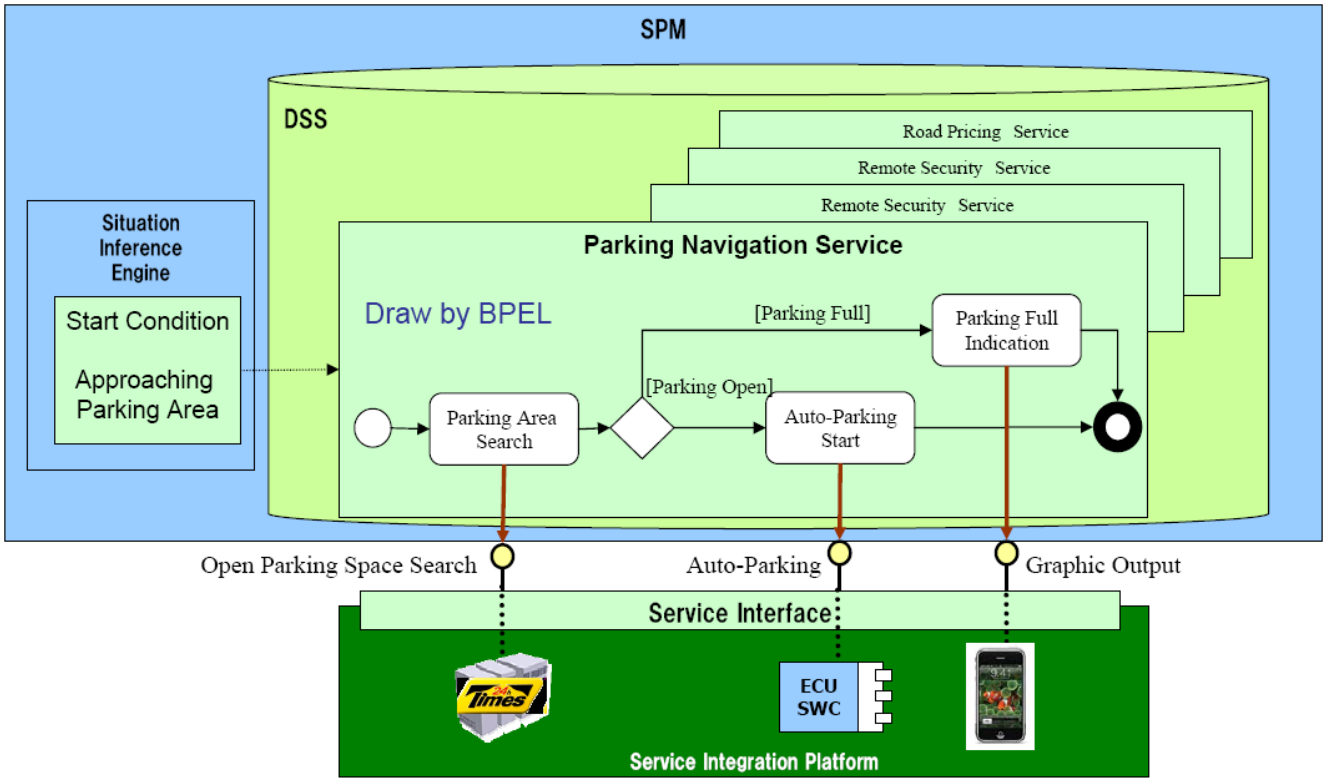

Figure 3. Easy Integration of services using Service Interface

4.2.1 *Resource Contract Function*

One of the key areas for us is to be able to see whether a service is appropriate, and to choose between multiple implementations of a service, depending on the context. For instance, some car navigation displays support 3D while others are only 2D. Even for the same car, the context may change, for example the available network bandwidth depends on location, service plan etc. A given service can thus offer multiple implementations, each tuned for a different context.

Trying to use the existing BPEL conditional statements to specify these choices would force us to model these decisions laboriously on a low level. It would also fix the method of choosing and the places where the choice can be made into the low-level models. Further, it would duplicate essentially similar decision mechanisms into many places.

Having tried unsuccessfully to store the necessary information in comments, we want to raise it to be a first class concept in our language. We call this concept the Resource Contract Function (RCF). It specifies the context in which this service implementation is valid.

Figure 4 shows the detailed process level specification of the Parking Area Search. Solid lines on arrows mean that the relationships are explicitly coded in scripts and/or conditions; dashed lines mean that the relationships are implicitly handled in the Service Process Manager (SPM). Numbered steps are:

1) During a navigation process the SPM tests the RCF start condition of a parking service process, to see if it is appropriate in the current context.
2) The start condition is satisfied, so the Parking Reservation Process is started.
3) This activity generates a new navigation script.
4) This activity suspends the current navigation script and starts the new generated parking script. Some data of the current script is handed off to the new script.
5) When the parking script finishes, the navigation script is resumed. Some data is handed off to the navigation script.

The Script Selection Condition RCFs allow the system to choose between two implementations of the service, picking the most appropriate one for the current hardware and network.

The actual running of the RCFs and the resulting choice is performed by the platform, freeing the individual models from having to specify this mechanism. This also makes the system more flexible for future evolution, and leaves more control in the hands of the car and its driver, rather than external services.

Currently we focus on the initial choice of the service and its implementation. Later we want to allow hot swapping to another implementation if a runtime resource becomes unavailable during process execution, invalidating the chosen RCF. In addition to the condition this would require moving the existing state of the current service to the new service: if we are part-way through parking, we do not want to have to start from scratch. One approach to this could be the explicit declaration of the core variables used in a service. The various alternative implementations of a given service would all use the same variables, making passing the state to the next service automatic (at least from the point of view of the modeler).

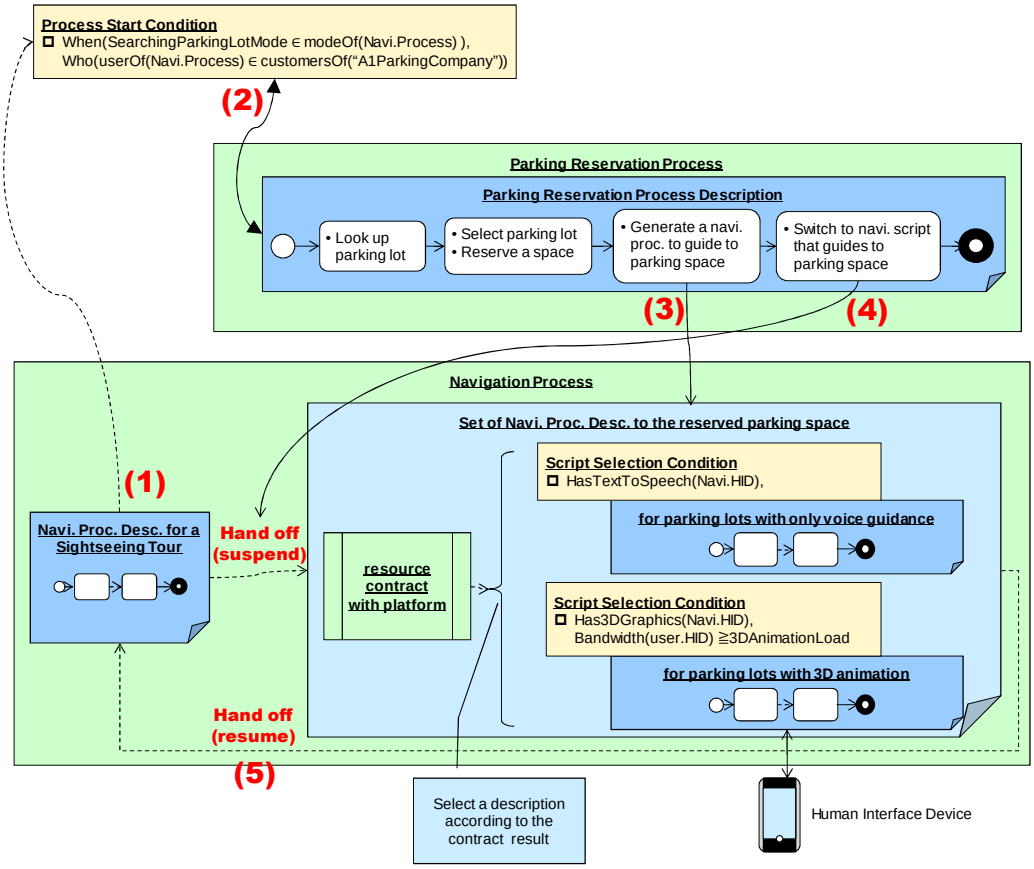

Figure 4. Process suspension and Resource Contract Function

4.2.2 Fault Tolerant Network

We have not yet created examples which model dependability issues such as reliability and security etc. Both of these are key concepts in our domain. For reliability we have a prototype design we call the Fault Tolerant Network. This area focuses on possible errors in execution, not just a resource becoming unavailable. It will require a separate process to monitor running service and spot and handle failures and degradation of service. The language will need concepts for specifying kinds of failure, counting failures, and how to resume, restart, hand off to a different implementation, or raise an error to the user.

5. SUMMARY AND FUTURE WORK

Due to service cooperation with information infrastructure, automotive electric systems are becoming increasingly large-scale and complex. Architecture approaches such as SOA, which cope better with integrating multiple elements from many partners, are becoming ever more important. Existing architectures in automotive only allow partial optimization in development, within a single company; we need to be able to move to total optimization, across all the companies in an emergent system.

We tried applying BPEL, the recommended technology for SOA in IT. At least in our case, we found that software technologies from IT industries cannot be applied unaltered. They must be improved and extended to better satisfy the particular requirements of the automotive domain. Attempts to extend them by "creative" repurposing of existing elements proved ineffective. A better solution, and one that would seem to actually require less work, is to create a Domain-Specific Modeling language of our own.

This experience is useful when applied to the broader questions of whether to use a standard language as-is, to improve its applicability but move it away from the standard by modifications specific to the current context, or to create a new language specifically for that context.

This is still research in progress for us: we do not yet have even a finished first draft of the DSM language, but we do have the requirements and know the basic elements. How best to structure those into a language, and how best to implement that language in a particular tool, remain to be seen. Our current thinking about such a domain specific language includes the following elements:

- **Virtual models of service elements in the real world**: the language should provide models of service existing in the real world, such as cars and parking areas. The language runtime environment corrects the gap between the status of the real world and virtual world, so that it keeps mirroring real instances in the virtual world. If it cannot maintain the consistency, it will notify errors to service processes.
- **Models of implicit synchronization of service processes**: the language should provide constraints specifying models that keep service processes running consistently. Although several service processes are developed independently by different vendors, such processes may support driving of the same car. In order for those processes to support driving consistently, those processes must run under some roles. The language should allow the models to describe such roles as constraints of process statuses.
- **Situation description models**: the language should provide situation description models which specify how to start or terminate service processes. We are currently investigating temporal logic as a suitable mechanism to track real world contexts and driving situations.

6. REFERENCES

[1] Aoyama, M., et al. 2008. Service-Oriented Architecture for Automotive Software Platforms, In *Proc. of 2008 Annual Congress (Fall)*, Society of Automotive Engineers Japan, No. 97-08, Oct. 2008, pp. 21-26 (In Japanese).

[2] Bocchi, L., Fiadeiro, J. L., and Lopes, A. 2008. Service-Oriented Modelling of Automotive Systems. In *Proceedings of the 2008 32nd Annual IEEE international Computer Software and Applications Conference*, IEEE Computer Society, Washington, DC, 1059-1064.

[3] Broy, M., Krüger, I. H., and Meisinger, M. 2006. *Automotive Software-Connected Services in Mobile Networks*, LNCS Vol. 4147, Springer.

[4] Erl, T. 2005. *Service-Oriented Architecture*, Prentice Hall.

[5] Iwai, A. 2009. Trends of automotive software platform, IPSJ-EMB, ICD2008-133, pp. 25-30 (In Japanese).

[6] Iwai, A. 2010. Designing and Evaluation of Automotive Service Integration Platform, *IPSJ SIG169*, Vol. 2010-SE-169 No.5 (In Japanese).

[7] Jordan, D., and Evdemon, J. (eds.) 2007. *Web Services Business Process Execution Language Version 2.0*, OASIS.

[8] Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., and Völkel, S. 2009. Design guidelines for domain specific languages. In *9th OOPSLA Workshop on Domain-Specific Modeling*. http://www.dsmforum.org/events/DSM09/Papers/Karsai.pdf

[9] Kelly, S., and Tolvanen, J-P., 2008. *Domain-Specific Modeling: Enabling Full Code Generation*, Wiley.

[10] MetaCase 2008. *MetaEdit+ Workbench 4.5 SR1 User's Guide*, http://www.metacase.com/support/45/manuals/