

Ontology-driven middleware for next-generation train backbones

Stijn Verstichel, Sofie Van Hoecke, Matthias Strobbe, Steven Van den Berghe,
Filip De Turck, Frederik Vermeulen, Piet Demeester
Department of Information Technology Ghent University, Ghent, Belgium
Gaston Crommenlaan, 8 bus 201
9050 Ghent
fax: +32 9 331 4800 - tel: +32 9 331 4981
stijn.verstichel@intec.ugent.be

Abstract

The current evolution in the railway domain heads in the direction of highly modular systems with many different smaller components working together, tuned towards the operator's needs. In order to create an integrated management system for the train and railway domain in general, many different applications from both operators and manufacturers have to cooperate.

To create a robust integrated system, a good supporting middleware infrastructure is needed. To the authors' knowledge, very few integrated or standardized techniques for creating middleware in the railway domain are defined to date. A distributed and modular architecture using ontologies, is detailed in this paper, providing the required intelligence needed for monitoring distributed applications in the train environment. The middleware allows information to be aggregated and analysed on different levels of the hierarchical architecture. Information querying, based on the ontology in the middleware is also discussed. By means of directory functionality, the ontology-driven middleware provides intelligent discovery as well. Finally, the ontology used in the middleware to structure the domain and corresponding methods for creating such an ontology are presented.

Keywords: ontology, middleware, distributed software, Web service.

1 Introduction

Recent research and development in the area of next generation train backbones has created an incentive towards the replacement of legacy interconnecting datacommunication architectures with newer and more innovative backbones. These new backbones both serve synchronous as well as asynchronous communication, using a middleware responsible for the management of the backbone. Because every train type is different and has its own operator-specific characteristics, the need for a modular architecture becomes clear. Moreover, not only the modular aspect of this architecture is important. Because the railway domain is a big and complicated one, especially if the entire domain is taken into ac-

count. This means that the description of the domain can rapidly become obscure and cluttered. The solution introduced here to overcome this inherent problem, is to create a hierarchical, layered ontology describing the domain. Using this approach, a well-organized and orderly description of the domain is constructed.

But first, before we introduce ontologies for the train backbone, an evolution overview of those backbones is given here. The starting situation is a collection of subsystems and the Train Control and Management System (TCMS) communicating with each other through several interconnecting wires. The next step was to introduce a single backbone for communication, either using the Internet Protocol (IP) or some other protocol. This approach had the disadvantage that all communication management and maintenance code had to be replicated in each subsystem, which was a time consuming and error-prone task. Therefore, the next step towards a more modular approach was the introduction of a middleware in the backbone. This middleware is responsible for all communication management processes in all its forms. These processes include life cycle management, discovery, transaction management, scheduling and so on.

One step further is now to introduce ontologies in the middleware. These ontologies create machine processable semantic knowledge of the domain. This means that the middleware now not only provides computational interfaces between subsystems, but also semantic interfaces. Extra knowledge about the subsystems is introduced into the middleware, creating a wide range of new facilities. Previous research in middleware architectures using ontologies is detailed in [1], where a service-oriented middleware for context-aware services is discussed. Also, research towards an easy integration and management of ontologies into applications and middleware is being done by the OMWG (Ontology Management Working Group) [2].

The remainder of this paper is structured as follows: Section 2 introduces ontologies and OWL by means of a brief railway situated example. Section 3 focuses on the detailed description of the constructed hierarchical layered software architecture. The developed communication network for the train backbone is presented as well. Also, the provisions for incorporating the ontology in the middleware is discussed. Section 4 introduces hierarchical ontologies for

the railway domain. Extendability and methods for adaptation of the ontology are presented in Section 5. Section 6 supports the ontology-driven management case by means of a use case about the detection of incipient door failures. Finally, we highlight the main conclusions and identify some future work.

2 Ontologies in the railway domain

2.1 Ontology definition

A brief, but all-embracing definition of an ontology, can be found in [3]: “An ontology is a specification of a conceptualisation in the context of knowledge sharing.” Accordingly, an ontology describes in a formal manner the concepts and relationships, existing in a particular system and using a machine-processable common vocabulary within a computerised system.

The foundation of ontologies is situated in the development of the Semantic Web. The way a search engine works these days is straightforward. It tries to match the keywords of a query with the content of the web pages. On these results statistical methods are applied, giving some results more relevance than other ones. An alternative to this basic search method, is to perform the search on the semantic concepts underlying the information on web-pages. Also, the relationships between web-pages are described by creating an ontology for the web-space. In such an ontology, the contents of web-pages are clearly explained and declared as well as the relationships between the web-page and other ones are outlined.

The main reason for creating an ontology of all the concepts within a domain, such as for example web-pages is that logical connections and relationships can be described between the concepts in the domain. This allows for inference to be applied on the ontology. The formalisation of, and the concepts in the domain themselves need to be chosen, as well as the relationships and logic that exist behind the concepts have to be specified as well. Once the ontology is constructed, inference rules can be declared about the concepts and their properties within the ontology.

Ontologies are certainly not only used in the context of the Semantic Web. Some examples of other domains where ontologies have proved useful are the creation of Location-Based Services or making applications context-aware. The use of ontologies to create context-aware applications is described in [6] and [7].

The concepts of ontologies are introduced here by means of an example of Passenger Information Systems (PIS). This example is modelled in Protege, an open source ontology modelling tool, developed by Stanford Medical Informatics [8]. Figure 1 shows the position of the PIS and some of the functions available in a hierarchical structure of the train. One can for example conclude that a “PIS” is a “Component”, which in its turn is a “RollingStockThing”. All concepts in the ontology inherit from the overall concept of “Thing”. Apart from the PIS as a device by itself, its associated functions are declared in the ontology as well. These

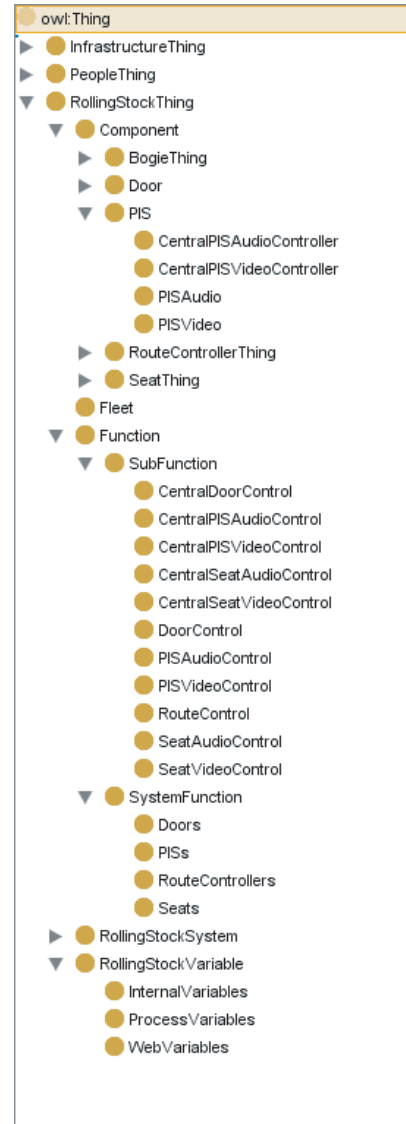


Figure 1: Hierarchical ontological conceptual structure situating “PIS” in the train

functions are defined as being either “SubFunctions” or “SystemFunctions”, corresponding to the scope of the function.

But, an ontology is more than just a hierarchical description of the domain. Relations between the several concepts are defined as well. Some of the relations defined for a “CentralPISVideoController” are “hasSubFunction”, “componentDefinedBy” or “isParentComponentOf”. One of the more specific function are “supportsMessageFormat” or “supportsMessageType”. These relationships declare the links between the controller on one side and its attributed functions or defining variables on the other side. All relationships defined for a “CentralPISVideoController” in this example are presented in Figure 2.

2.2 Ontology Web Language

OWL consists of three sublanguages, each of them varying in their trade-off between expressiveness and inferential complexity. They are, in order of increasing expressiveness:

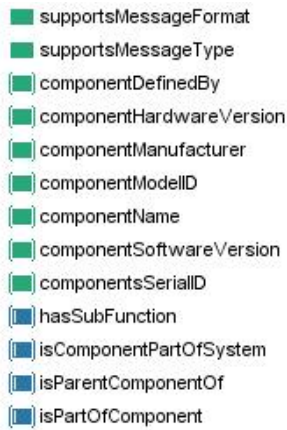


Figure 2: Representation of the relationships of a “CentralPisVideoController”

- OWL Lite: supports classification hierarchies and simple constraint features.
- OWL DL: OWL Description Logics, a subset providing great expressiveness without losing computational completeness and decidability.
- OWL Full: supports maximum expressiveness and syntactic freedom however without computational guarantees.

The syntax of OWL is based on XML (eXtensible Markup Language), the formal foundation for its semantics is based on Description Logics. OWL is the natural evolution of several previous WC3 recommendations, being XML, XML Schema, RDF and RDF Schema.

Using one of the three sublanguage flavours of OWL, one can easily adapt to the required expressiveness. Arguably the most interesting sublanguage for many application domains is OWL DL, balancing great expressiveness with inferential efficiency. The efficiency is guaranteed by the underlying Description Logics. Due to its foundation in Description Logics, OWL DL is also very flexible and computationally complete. Ontologies are considered as dynamic and evolving in time. As ontologies are also tailored towards the distributed nature of the Web, OWL additionally provides constructs for (de)composition, extension, adaptation, sharing and reuse.

The ontology briefly introduced earlier is easily extendable if new types of “Functions” or new types of “Components” are introduced. As an example, a new type of “Function” called an “InCaseOfEmergency” function is introduced. This function defines the actions to be taken when an emergency occurs. New relations between the existing “Components” and the new “Function” are declared, so that the ontology is now extended with some new semantic description.

2.3 Querying ontologies

The use of ontologies opens up a wide range of new querying facilities. Looking for concepts in an ontology can

be performed in a number of ways like the exact lookup of concepts using their name, but also querying based on properties or characteristics is possible. Using the reasoning that can be performed on OWL DL, supported by its first order logic, concepts can be derived and found that match the properties specified in the query. For example, they allow checking whether objects are consistent with concepts, retrieving all objects that are instances of a particular concept, or retrieving all objects that satisfy certain conditions.

Mapping this on the example of the PIS, one might to look for all “Components” that have an associated “SubFunction” of “PISVideoControl”, in order to display some information of the next stop of the train. Another example is the query for all “Components” that have an outdated firmware version. This relationship is defined in the ontology by the relationship “hasHardwareVersion”.

2.4 Discovery

Using the above methods for querying, more intelligent service discovery mechanisms in a Service Oriented Architecture (SOA) are possible. Currently most registries use a simple name to reference resolution. By using the querying methods above, a lookup for services can be performed using a specification of characteristics and properties. The reasoner behind the registry-ontology can then infer the correct, or the most appropriate reference towards a service.

3 Management infrastructure for the next generation train backbones

Ontology-driven management aims to provide intelligence and reasoning inside the middleware itself, instead of having this intelligence in the application layer. Information provided by several agents, is aggregated in an intelligent way and correlated by the ontology-driven middleware in order to provide richer semantics to the users of the middleware. Also, requests from clients to the middleware are handled in a similar intelligent manner. A single request from a client, might then be broken up into several sub-requests by the ontology-driven middleware, providing a richer functionality to the requestors, whilst being transparent. A final functionality provided by the middleware, is that of a directory. Using reasoning performed on ontologies, composition of several agents into a larger virtual agent is supported and discovery of agents based on particular characteristics is possible.

Middleware enables and simplifies the integration of components developed by multiple technology suppliers. Middleware, such as CORBA, DCOM, Java RMI or Web services, makes it easy and straightforward to connect separate pieces of software together, largely independent of their location, connection mechanism and technology used to develop them. CORBA and Web services have obtained success due to their standardised interoperability and compatibility with other languages. Contrary to other middleware, CORBA and Web services are open, platform and vendor-independent. Performance of both middleware platforms has

already been investigated by the authors and reported in [4]. Results show that CORBA is more performant. Web services lose some performance in favour of an easy learning curve. Because of the performance benefits and several extra facilities and functionality readily provided, the CORBA middleware was chosen for the development of a next generation train backbone. These extra services include e.g. Directory Services, Event Services or Transaction Services. The choice for CORBA as the preferred middleware was also strengthened by the fact that CORBA is a mature technology and has already proven its capabilities in many other projects.

The CORBA-based system that has been developed to provide the functionality, needed to satisfy the requirements of this highly modular and distributed environment is explained in greater detail in this section. This architecture serves as a basis for the development of an ontology-driven middleware.

3.1 Architecture for the train backbone

The complete architecture, developed for this system, is a hierarchical, layered one. If the structure of the railway domain is examined more closely, a distinction can be made between a number of subsystems, each of them providing some well defined functions in the entire overall railway system.

Probably the most intuitive illustration would be to look at a fleet of trains. A particular fleet consists of a number of trains belonging to the fleet. Each of those trains is in its turn composed of one or more coaches. This, however, is not the bottom layer of the hierarchy. The decomposition can be driven even further, as each coach has a number of subsystems, e.g. doors, lighting, airconditioning or the earlier quoted Passenger Information System.

The architecture designed for this system, must therefore facilitate this inherent nature of layers. Each of the individual layers identified is represented by a software layer in the train backbone architecture. The different layers are interconnected through gateways. On the one hand the gateway serves purely as a communication interconnection between the layers. On the other hand, intelligence is inserted into the gateway as well, which is examined more closely in this section.

The overall interconnecting communication network is presented in Figure 3. Using a bottom-up approach towards identifying all the layers, the subsystems in the coaches are interconnected at coach level using a coach-level network. Each subsystem has a gateway towards this network. In this way connectivity between the subsystems is possible, and using CORBA, both synchronous as well as asynchronous communication is supported.

Now that communication facilities are provided between all subsystems in a single coach, communication throughout the entire train has to be set up, again using a gateway responsible for the entire coach it is located in. As can be seen from Figure 3, all gateways of the coaches are interconnected on train-level, using a train-level network.

Trains are dynamic environments; trains couple and decouple many times during the day, resulting in a change of

network topology every time. Along with the changes in topology, the location and the type of available services on board alters as well. The middleware provides functionality towards the management of the services on board.

- Automatic configuration agents are located in every coach. They have the responsibility for setting up and maintaining the train network with the correct configuration. These agents are mainly responsible for checking the consistency of the underlying communication network, in our case the IP network, and alter the configuration settings accordingly.
- On board every coach, a CORBA Naming Service is located. This service is responsible for listing all services on board its own coach. It typically answers name to reference lookups from clients, wishing to establish a collaboration with another service.
- Another aspect in the architecture is the provision of asynchronous event-based communication. To make this possible CORBA Event Services are constructed. The gateways have a major responsibility in this mechanism as well. They forward events in both directions between the coach level and train level network if necessary.

3.2 Intelligence incorporated in the backbone

One can clearly see that it is not necessary to distribute every piece of information through all the layers in the system. Therefore, each layer performs some analysis on the data provided by the monitoring agents in its own layer. Taking the lights subsystem as an example, it would be unnecessary to let all other layers know if a single light bulb is defective in a particular coach when that coach is still lightened by other light bulbs. If, as a result of this failure, the entire coach is covered in darkness, then notification by the gateway towards other layers is necessary. Moreover, the gateway also additionally performs analysis tasks such as filtering. If events pass through the train-level network, not being of any use for the coach the gateway is responsible for, then it is unnecessary to allow this event to enter that particular coach layer, and vice versa.

Also, aggregation is a necessary function to be supported by the gateway. A single failure of some system, might result in several other systems failing as well. In order to optimize fault diagnosis, all faults are collected, correlated and transmitted as a single notification to the other layers.

3.3 Ontology processing in the middleware

In order to provide functionality for the integration of the ontology into the middleware some extra ontology processing services are provided in the software architecture for the train backbone.

The construction of the ontology is presented in section 4, but the basic idea for the construction of the train ontology

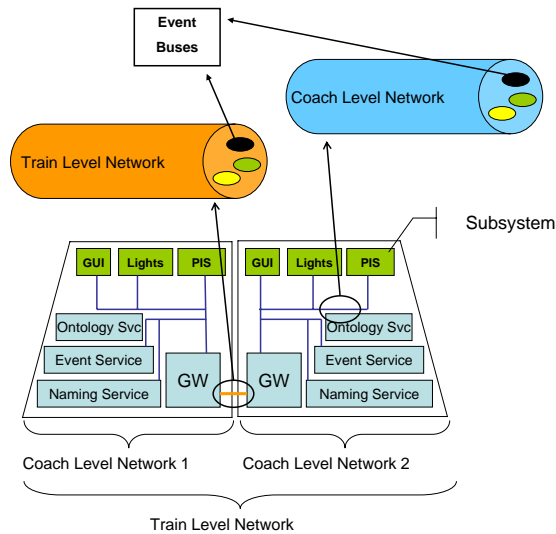


Figure 3: Hierarchical layered train network with its ontology extension

and the software architecture for the train backbone is rather similar. They are in the fact both created using a layered and hierarchical approach. Therefore, the natural way to combine the two is to map the layers of the ontology with those in the software architecture for the train backbone. This way, an integrated middleware is constructed, providing a rich functionality towards the subsystems in the train.

An ontology is not useful by itself, that is, if there is not a service available to process the ontology and to infer extra knowledge about the domain. Also queries for information, based on the ontology, have to be answered by a software component in the architecture. The extension of the initial architecture is also represented in Figure 3.

We introduced new middleware services at the same hierarchical levels as the other middleware services, such as “Naming Services” and “Event Services”. This means that “Ontology Processing Services” are introduced both at coach-level and train-level. These services can then be used by other services or subsystems if they need richer semantic knowledge about the domain.

Using this separation of concerns, a delegation of responsibilities is facilitated. To illustrate this statement, the example of the train lighting can be used again. A particular service might only be interested to know the effects and inferred actions needed on coach-level when a defective lighting is found. In order to get this information, the particular service only has to contact the “Ontology Processing Service” on the coach-level network in its proper coach. On the other hand, if train-wide actions have to be known for some reason, this is done through the “Ontology Processing Service” on train-level. After all, it is this train-wide service that uses the train-wide ontology to execute its inferential algorithms on and therefore has a train-wide knowledge base to use. How these ontologies are created is now discussed in the next section.

4 Hierarchical ontologies for the railway domain

Because ontologies are derived from the Semantic Web, they form the ideal basis for modelling in distributed environments. Individual ontologies can easily be shared, extended, adapted and composed into larger ontologies. Therefore new ontologies can be created, combining existing ontologies and newly developed ones. Thus, it is the ideal method to create knowledge bases for the dynamic distributed train environment. Therefore we introduce here hierarchical ontologies for the physical description of trains and for the network infrastructure in this environment.

The hierarchical layered approach towards train backbones has already been discussed in section 3. The next step is to combine this hierarchical layered approach with the ontology concepts introduced in section 2. Ontologies form a crucial part in developing a common understanding of the domain. But, at the same time, the pursuit to have a common understanding, shared by all the actors and subsystems brings along major challenges, especially in a large domain like the railways.

The basic building blocks, forming the lowest layer in the hierarchical ontology, are the subsystems such as doors, lighting, passenger information systems and airconditioning. In this way, those subsystems are the first entities to have ontologies. Of course, these subsystems don’t exist only by themselves. They are located in some particular coach and cooperate throughout the normal functioning of the train. So, the next level in the ontology is the coach-level ontology.

This ontology describes an entire coach, using the ontology provided by the composing subsystems. The various ontologies are thus combined to form a larger, integrated ontology. This ontology then serves as a common understanding of the coach and its internal working. Analysis tools for fault analysis in the coach, as an example, can therefore use this ontology, which includes the detailed description of the composing subsystems, to execute their algorithms on. Also, procedures can be specified in the ontology for event-handling. Some events are processed locally in the subsystem or in the coach itself, others need to be transmitted towards the overlying train-level layer in the hierarchy.

Indeed, by going one step higher, an ontology is created on train level as well. Again, this ontology is dynamically composed from all individual coach level ontologies, resulting in a general, common understanding of the train as a whole. Analogous to the coach level analysis tools, train level analysis tools can use this new high level train-wide ontology as a description of the entire train, taking into account the information provided by the underlying ontology. A detailed use case of fault-analysis on the train is presented in section 6. Since trains are often not homogeneous sets of coaches, the coach level ontology is used as a layer in between the ontology describing the individual subsystems and the overall train-wide ontology. For example, a particular train can consist of a locomotive and several coaches, not necessarily all of the same type. Another example are the so-called, “Multiple Units”, consisting of a number of differ-

ent types of coaches, classified according to the functionality provided. Examples are trailers, driving van trailers or motored coaches. By creating an ontology in between we also have local knowledge of the coach, creating a point of redundancy.

Apart from the description of the physical composition of the train, an ontology can also be used to create an understanding of how communication can be established between subcomponents on the train. The ontology then describes how the different services and subsystems interoperate, and how they can be contacted. Also, the characteristics of all composing subsystems can be specified in the ontology. This could incorporate not only the interfaces of the services and their attached functionalities or dependencies, but also e.g. the required bandwidth needed to perform a certain request.

The characteristics of the entire communication network can also be included in the ontology. The different data flows with their respective needed amount of resources is another aspect of communication that can be defined in the ontology, resulting in the ability to automatically infer affected services and functionalities in case of some link or other failure. By using an ontology for modelling the data flows and their characteristics, a clear understanding of available and used resources of the communication network is provided, and moreover in the same way as the other descriptions of the subsystems. The advantage of this common approach is that one single modelling method is used to provide a description of the entire domain, including both the physical representation and communication infrastructure with its available services and dataflows, throughout all the different subdomains. Therefore the aim to create one single, common understanding of the domain with all its characteristics has been fulfilled.

5 Extendibility and adaptation of the ontology

Starting from the basic building blocks of the individual subsystems, we composed an overall hierarchical train-based ontology and corresponding processing services in the middleware. This section discusses why this approach was taken and gives an example of the need for this bottom up approach.

It would be a huge task to create one single ontology, useful in the entire railway domain, certainly when the infrastructure is also taken into account. Illustrating the complexity of the domain, the concept of a train door is oftentimes used. At first sight, this subsystem on-board of the train, is very simple; it can open, close, lock and unlock.

Extending this initial trivial description of the door, the difficulty is found in the lower level representation of the door. Every manufacturer uses different subcomponents to construct a train door, having a repercussion in the ontology. Using the initial thought of creating a single ontology for the entire railway domain, it would be unthinkable and unfeasible to model every type of door in that ontology.

This can be overcome by developing a skeleton ontology model for the domain. In this model the major subsystems are identified, but the exact detailed representation is

not constructed in this skeleton. Making use of the facilities provided by OWL, being the support for distributed extension of the ontology, the manufacturers can themselves provide an ontological representation of the door. This ontology is then inserted into the overall ontology for that particular train. Also, alterations to the internal working of the door, either to a whole fleet, or just to a particular train, can be reflected in the ontology description of that particular door.

6 Use Case details

One of the main incentives for creating an ontology-driven middleware is the possibility it creates for intelligent querying of the domain. To illustrate this, a use case is presented here, demonstrating some of the strong points of this approach.

A key factor in the performance indicators of a train is its punctuality. This punctuality is to some extent influenced by the door closing times. Many of the delays experienced in the timetable are due to slow door closing, or doors failing to close along the route. Many of these failures could be prevented using the ontology-driven middleware.

Not all types of doors are the same, one can have, for example, pneumatically operated or electrically operated sliding doors. It is however almost impossible to build a generic fault-analysis tool for all types of doors at the same time. Each of these types has a very different internal working and draws its power from very different sources. Using an ontology describing these doors, the story is completely different. The intelligence is now not in the fault-analysis tool itself, but in the middleware. Therefore, a generic fault-analysis tool, using the inherent knowledge of the middleware, can now more easily be built.

This use case deals with the detection of incipient door-failures. Often an upcoming failure can be detected through an increase in the power consumed by that door. Of course, this only applies to electrically operated doors. On the other hand, air leaks can be identified as an indicator of an upcoming failure of a pneumatically operated train door. This difference clearly indicates again, that it is almost impossible to create a single ontology, since these examples are only two of many types of doors, unless the manufacturers provide with their hardware the ontology describing it.

The manufacturers of the doors should normally have a clear understanding of the working of their doors. They should therefore also know about some weaker points in the design or possible indicators for incipient failures. By including this information in the ontology, the middleware is able to identify the doors that need attention, even before the door actually fails, using the inference methods applied by the reasoner on the ontology.

But this shouldn't necessarily be the end. Other railway experts, such as mechanical engineers and operators, might also want to insert some additional knowledge in the ontology. An example of this could be the inclusion of a rule defining that if all doors on a train are closing too slowly, one doesn't have to look for door-faults, but the likelihood of something else failing is very high. One of the reasons could

be the power unit failing, or in the other case, the compressor being unable to create sufficient air pressure. These rules are likely to be inserted and established by the experience of staff working on the trains.

The examples described above all use the classical request/response mechanism. It is the operator asking for incipient door-faults. On the other hand, thresholds for certain property values can also be inserted in the ontology. The middleware could include some event handling functionality, monitoring the values of the defined properties. Rules can therefore be inserted in the ontology, describing which components are affected by the out-of-range value of some property monitored.

Different types of events can be included in the rules, according to their level of urgency. These events can then for example be logged into a fault-description database or directly be transmitted to the operator. In the first case, the database is consulted by depot engineers when the train enters the depot for servicing, so that incipient faults can be repaired before the actual failure occurs. In the latter case, the fault is of a higher urgency, and should be rectified as soon as possible. In case the fault has already occurred, the train should be taken out of service, because the safety of passengers and staff might not be guaranteed anymore.

This functionality can also be looked at from another point of view, namely as that of a virtual door-fault analysis service. Requestors ask this virtual service for all doors failing on a given train, but in fact, it is the middleware composing several smaller sub-services, such as all the doors, the power supply unit and perhaps other ones. All those smaller subsystems compose the larger virtual service. This is another great functionality provided by the ontology-driven middleware.

7 Conclusion

More and more, the railway domain is evolving into a technically high-profile environment. In addition, software and intelligence is inserted into the systems of the railway domain. Consequently the need for intelligent distributed applications increases. An ontology, describing the domain and the services, could prove the ideal way to insert this intelligence in the middleware. Reasoners can then infer some extra knowledge about this domain and expose it to the users of the middleware.

This paper introduced a solution for integrating the intelligence into the middleware of a distributed system. More specifically, the example of the railway domain was taken to demonstrate the ideas behind this approach. The advantages of using an ontology to describe the subsystems in this domain were presented as well.

The ontology-driven middleware provides a solution for the need for intelligence. It can be used to develop generic and lightweight applications. These applications can use the middleware and its inherent intelligence to acquire the necessary knowledge about the domain and fulfill their desired function. Moreover, the ontology-driven middleware provides intelligent discovery of services or objects, as well as

service or object composition. In the latter case, the middleware provides a virtual service to the outside world, but inside, this virtual service is composed of several smaller real services.

Using the extendibility of an ontology, manufacturers can provide the railway domain with the specific knowledge about their produced subsystems. This greatly reduces the effort needed to create an ontology for every train, since the ontology would then be created combining the individual ontologies of all composing subsystems on that train.

We also discussed the methods for including this ontology in the middleware and introduced new functionality for the middleware, since the use of ontologies opens up a wide range of new possibilities. To conclude the paper, an example use case of a door-fault and incipient door-fault analysis was presented.

Acknowledgement

We would like to acknowledge that part of this research has been supported and incorporated into Integrail, a project in the scope of the European Commission's 6th Framework, in the thematic area of sustainable surface transport development.

References

- [1] Gu, T., Pung, H. K., Zhang, D. Q.: *A service-oriented middleware for building context-aware services.*, Journal of Network and Computer Applications (JNCA), 28(1): 1-18, January 2005.
- [2] OMWG: *Ontology Management Working Group*, <http://www.omwg.org/>.
- [3] Gruber, T. R.: *A translation approach to portable ontology specifications.* Knowledge Acquisition, 1993.
- [4] S. Van Hoecke, T. Verdickt, B. Dhoedt, F. Gielen, P. De-meester: *Modelling the performance of the Web Service platform using Layered Queuing Networks*, The 2005 International Conference on Software Engineering Research and Practice (SERP'05), Las Vegas, USA, 2005.
- [5] McGuinness, Deborah L. and Frank van Harmelen: *OWL Web Ontology Language Overview*. W3C Recommendation, 2004, <http://www.w3.org/TR/owl-features/>.
- [6] Gu, T., Pung, H. K., Zhang, D. Q.: *Towards an OSGi-Based Infrastructure for Context-Aware Applications in Smart Homes.*, IEEE Pervasive Computing, 3(4), 2004.
- [7] Preuveneers D., Van den Bergh J., Wagelaar D., Georges A., Rigole P., Clerckx T., Berbers Y., Coninx K., Jonckers V., De Bosschere K.: *Towards an Extensible Context Ontology for Ambient Intelligence.*, EUSAI 2004: 148-159.
- [8] Protege: *A free, open source ontology editor and knowledge-base framework*, 2005, <http://protege.stanford.edu/>.