

Figure 2. A UML class diagram detailing the generic neighbourhood information model. The two possible NIM-Components are shown as well as nesting possibility of components to enable hierarchy.

with storing. The addition of an expiry date for each value can be used to delete stored values after a certain point in time. Since historical data can also be queried via the data model this is especially helpful, because it enables that values can be padded automatically and that users or systems can specify the lifetime of the stored values. Another security mechanism is added by additional information to determine the agreed usage. The field itself contains information on all other people, roles, service identifier, etc. that are allowed to access the entry. Since a list of all people or names that might access the entries is unfeasible the NIM can be used by a system using role based access control (RBAC) concepts. The role for users or a group of user can be added to the agreed usage data field of an entry.

The last security aspect is the specification of the geographical, physical location where entries or values are allowed to be store. This information is necessary since most countries have different laws on privacy and users may want to decide in which countries or geographical locations entries and values of the NIM should be stored. The information can then be used by an implementation to determine the storage location. Moreover this information can serve as a decision base for users when deciding which information should be uploaded to a platform which uses the NIM as data format.

3.3 Implementation of the Neighbourhood Information Model

When technically implementing such a NIM on a platform additional aspects have to be taken into account. Especially the frequency of new values has to be considered. Automated and regularly measured values which are entered into such a platform can

lead to a high data load which has to be handled in a different way than structural data of the neighbourhood which only changes sporadically over time. This can be achieved by e.g. only sending messages when there is a change in the value so that the last sent message value remains the current or real-time value. There are several reasons for this decision, but the main reasons should be performance and efficient use of storage capacity.

Also other important non-functional requirements have to be taken into account. Since two different neighbourhoods mostly do not have exactly the same data models, and future neighbourhoods may need data which cannot be foreseen at the moment or a single neighbourhood may evolve over time and thus, the corresponding data model has to be adapted, an adaptable and extensible implementation of the NIM is necessary.

To provide an adaptable and extensible implementation a mechanism for storing unforeseen entries that may arise during runtime of the overall systems is necessary. Therefore the prototype makes use of the generic meta model as its internal domain model which allows for the handling and persistence of arbitrary data entries fitting the meta model. Nevertheless the entries from a different data model have to be transformed into the meta model to be stored in the database. However this approach has the drawback that it works on the basis of a very general domain model which is uncomfortable to use. Moreover, data entries from other data models have to be transformed into the meta model format which would lead to enormous effort if implemented manually for each existing data model. But even if such a transformation would available via a manual implementation, the extension at runtime would not be possible for the new NIM entries. To overcome

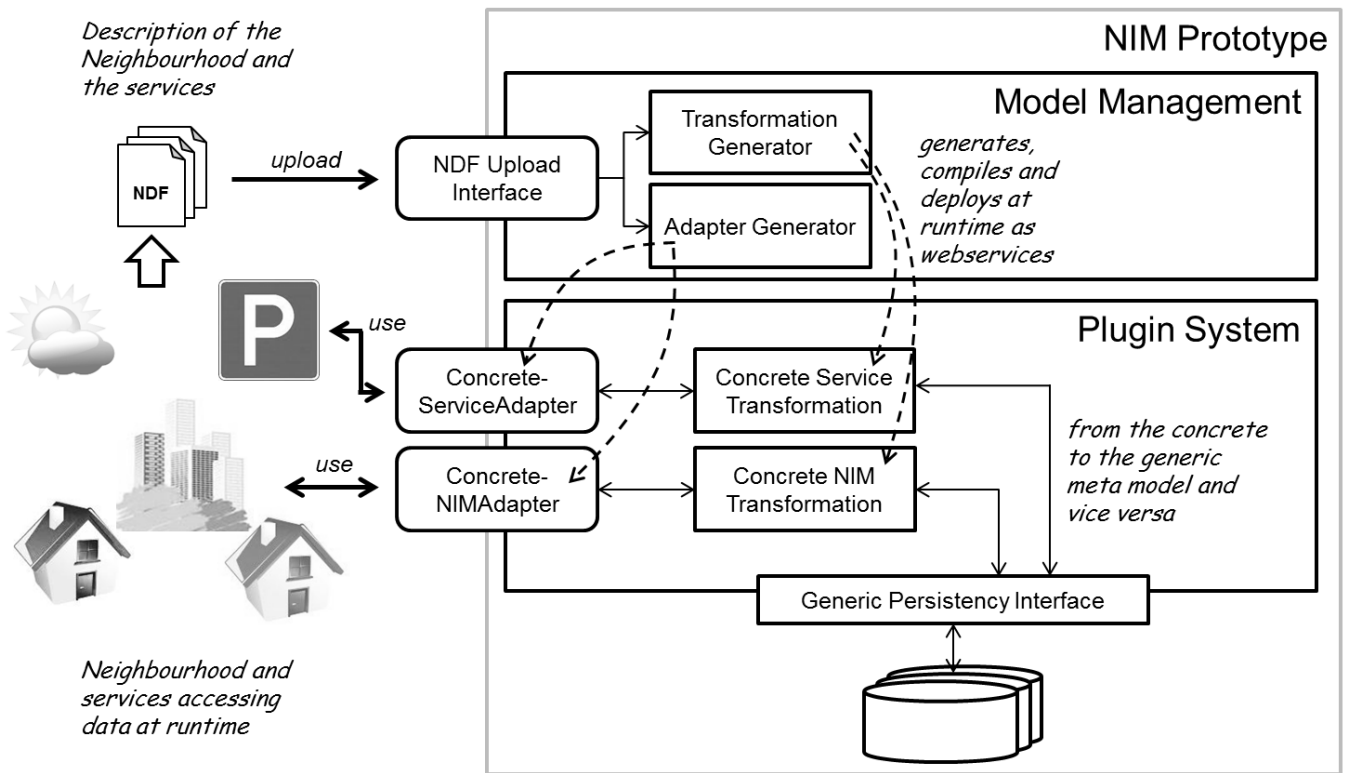


Figure 3. Overview of the overall implementation showing the Model Management component responsible for generating the adapters and the transformations, as well as the Plugin Systems being the runtime environment of the adapters.

these problems we make use of a plugin based architecture that allows us to add new plugins at runtime. The plugins themselves are generated by a code generator that processes our input models at runtime. The generated code includes code responsible for the transformation between data in the concrete and generic format. In this way both problems have been solved.

The models are written in a DSL invented for describing concrete NIM data models in a NIM data format (NDF). It should be noted that the NDF specifies a data model and not concrete data. The methodology to configure the running prototype is shown in Figure 3.

The implementation consists of two components: a Model Management component and the Plugin System. The Model Management component is responsible to generate new adapters and transformation code if a new model is uploaded. The Plugin System itself represents the overall system that is used by services, and neighbourhood elements for accessing the data. To connect a new neighbourhood or a single service to the platform three steps are necessary.

In the first step an NDF model, specifying the concrete NIM format, has to be written in a textual syntax and uploaded to the prototype. The Model Management component analyses the NDF, checks context conditions and if it is valid, the component internally passes the model to the transformation generator and the adapter generator which generate a concrete adapter and transformation services. Both

parts are then deployed to the Plugin System automatically. It should be mentioned that both the code generation and the deployment of the adapter and the transformation are performed at runtime of the system. The adapter serves as a webservice endpoint and offers an interface related to the concrete data model. It enables storing and retrieving the concrete data types. The transformation contains the logic on how data from a data model can be transformed to the generic data model and vice versa, which will be discussed after giving an example of the NDF.

The second step is then the usage of the new adapter by neighbourhood services operating on the defined data Model. The NDF models allow specifying the structure of neighbourhood information data. For example we could specify that rooms have a room name like this:

```
Room {
  String roomName;
}
```

This example clarifies that the NDF specifies a data format and not the concrete data, since we specified that each room has a room name, but not how many rooms exist and how the concrete names of the rooms are.

Since a single NDF file for the whole data model may become unfeasible and different neighbourhood services may need a distinct set of data entries of the concrete NIM, the definition of several data models in several NDFs is possible. Thus also different

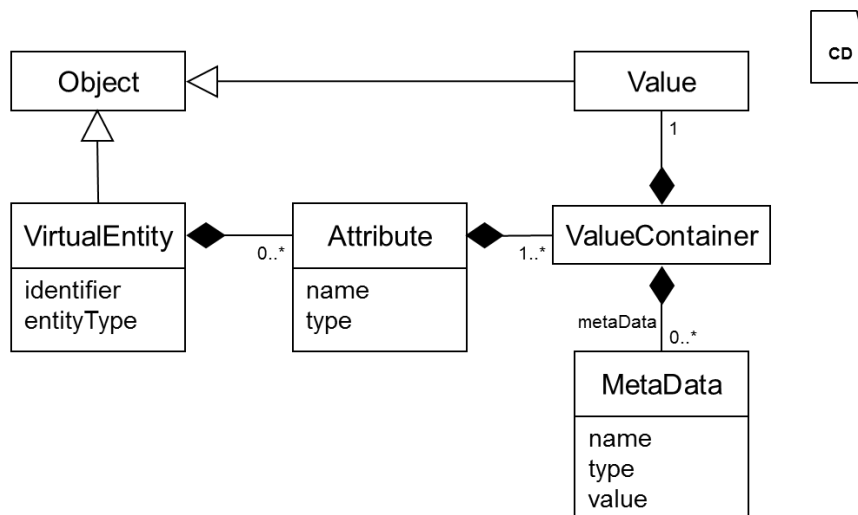


Figure 4. The edited domain model of the Internet of Things (IoT) approach (IoT.est, 2013)

stakeholders as well as specific service developer may develop their own description. Consider two different buildings with different data models, should be connected to the Neighbourhood Information System. In this case a different definition of a room, for each building can be provided. In combination with the previous example such a second extension could look like:

```

AnotherRoom {
  String roomID;
  Number surface;
}
  
```

Regarding the transformation between the concrete data model and the generic one we have to consider several cases. The first case, transforming a concrete data model into the generic data model is relatively straight forward. Taking up the example of the *Room*, this would become a category with name “room” within the generic data model. Additionally the category room would have an entry with name *roomName*. Since the NDF only describes data types and no instances the values of the entry are allocated at runtime and would contain the actual room name. Furthermore at runtime there would possibly be several instances of a room. The DSL also allows for hierarchically defining data types within other data types. This can be reflected via the category composite in the generic data model.

The second case is the transformation from the generic data model back to the concrete data model. If the data stored in the generic model originates from the same concrete model it should now be transformed back to, the transformation is as straight forward as the previously explained transformation to the generic data model.

The third case is the transformation of data originating from a different concrete model than it should be now transformed to. It is most likely that this is the most common case since every service and

neighbourhood element wants to access data from other systems and has to know their data model therefore. Assuming that there may be an additional service which needs information about all buildings independent of the format, there are two possibilities:

- Using all the required existing adapters separately
- Specifying an additional NDF with a mapping included

Of course a service implementation can use every existing adapter separately and aggregate the data manually from the different formats. Nevertheless this is not envisioned and unfeasible.

The second alternative is more comfortable. The service developer can define the data in his own format like:

```

StandardRoom {
  String identifier;
}
  
```

In addition he needs to specify the mapping between his own room type and the already existing types:

```

StandardRoom.identifier := Room.roomName /
  AnotherRoom.roomID;
  
```

This type mapping information can be added in addition to the format definition in the same NDF model file. Note that the data type name serves as a namespace and the included field can be directly accessed. The NDF also contains a package declaration concept, not illustrated here, that enlarges the possible namespace and thus name clashes are to be considered unlikely. The list of values that are used for the identifier is separated by “|” and the list elements reference specific field from other NDFs. Additionally context condition checks ensure that the referenced types exist. On a technical level this is achieved by the language integration concepts of-

ferred by MontiCore (Völkel, 2011) (Look, et al., 2012). The generated adapter of our implementation takes care of the aggregation and offers the following methods to access the new defined Data:

```
List<StandardRoom> getAllStandardRooms();
```

This method allows resolving the data of all rooms, automatically converted in the standard room format.

4 RELATED WORK

In this section we provide work related to the NIM and to the integration of heterogeneous data sources.

In general there are two types of related work to be considered: Those following a standardization approach and those following a generic approach. Both differ in the way of integrating different heterogeneous data sources. Even within the two approaches there are different techniques applied for matching one data model to another. This can be done by ontologies, link models or adaptation.

Additionally approaches focusing on static data provided by information models and approaches focusing on measured, calculated building management systems (BMS) data have to be distinguished. Approaches like the OPC Unified Architecture (OPC UA), as an IEC standard (IEC 62541) (International Electrotechnical Commission, 2008), already provide an abstraction from the various management systems and will not be discussed in more detail. Nevertheless for static data there exist approaches such as the Industry Foundation Classes (IFC), registered under ISO-16739 (buildingSMART International Limited, 2013), the Building Information Model (BIM) (Eastman, et al., 2011), CityGML (Open Geospatial Consortium, 2007) and gbXML (gbXML, 2013), just to name a few. While most of these approaches stop at the building envelope they also do not provide extension points for including additional data. Within CityGML it is possible to define extensions but these extensions also rely on a generic model. A detailed discussion of the advantages and disadvantages can be found within the Adapt4EE project (Tzovaras, et al., 2012).

Approaches like the HESMOS project (Hesmos Project, 2012) are trying to setup an integration platform that can integrate different data models via link models (Liebich, et al., 2011). These link models basically serve as adapters between the different data types.

Within the eeSemantics Community (eeSemantics, 2012) there are several research projects aiming at solving this issue. Approaches, such as the SEMANCO project (Semanco project, 2012) aim at defining a standardized data model as ontolo-

gy, which is able to capture most of the data occurring in a building or in a neighbourhood.

Ontologies are commonly used when deducing new information from existing data models or when defining common concepts or mappings between data models. The main difference to our approach is that ontologies are mainly applied between two concrete data models and are thus bound to the concepts provided by the models.

Within the SEMANCO project a comprehensive list of possible data has been identified. The community also aims at finding ontologies for other areas apart from building or neighbourhoods. Thus there exists the FIPA device ontology (Foundation for Intelligent Physical Agents, 2002) and several smaller ontologies created within the AMIGO project (Georgantas, 2006). There are also concrete approaches considering sensor ontologies (Barnaghi, et al., 2012), linked data (Le Phuoc, et al., 2011) and sensors as a service (Compton, M., et al., 2012) existent. Nevertheless, following a service oriented approach within the neighbourhood, this fixed data model is too rigid to be able to capture newly arising data created by services running within the neighbourhood. The underlying data model has to be changed if new information does not fit the current model. Thus our approach relies on a meta model backend and a mapping that is quite similar to ontology mapping approaches (Euzenat & Shvaiko, 2007). Other approaches like the Internet of Things approach (IoT-A) (IoT.est, 2013) (OpenIoT, 2013) (Probe-IT, 2013) already rely on a generic data model, shown in Figure 4.

Our Implementation of the NIM is close to the IoT-A model where we chose to have no explicit virtual elements and no explicit metadata. Nevertheless our model is not restricted by this.

5 CONCLUSION

Within this paper we have presented our approach to integrating heterogeneous building and periphery data models into a neighbourhood information model. We have shown how the mapping from a concrete manifestation of a data model to the underlying data model can be achieved and how the domain expert is enabled to express domain knowledge in a DSL which in turn is used to fully generate the required code at runtime of the overall system.

Nevertheless for the future we envision more work on the tooling infrastructure. At the moment the service developer has to know the existing data models and has to extend the mapping if new data models are integrated into the system. We envision this to be resolved by a tooling infrastructure providing access to a library of existing models.

Moreover the mapping defined between different models does not allow to distinct between specific

instances. It is not possible to receive the data from specific instance in the standardized format, yet. For now, the information of all available instances matching the mapping instruction is resolved.

6 ACKNOWLEDGEMENTS

The work presented in this paper has been carried out in the COOPERaTE project, co-funded by the European Commission within the 7th Framework Programme (FP7/2007-2013) under grant agreement no 600063.

7 REFERENCES

- buildingSMART International Limited, 2013. *IFC 4 Documentation*. [Online] Available at: <http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm>
- Barnaghi, P., Wang, W., Henson, C. & Taylor, K., 2012. Semantics for the Internet of Things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS)*, pp. 1-21.
- Compton, M., et al., 2012. The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. *Journal of Web Semantics*.
- COOPERaTE, 2013. [Online] Available at: <http://www.cooperate-fp7.eu/> [Zugriff am 10 April 2014].
- Corrado, V. & Ballarini, I., 2012. *D3.1 Report on the Accessible Energy Data. SEMANCO Semantic Tools for Carbon Reduction in Urban Planning, Project ICT 287534*, http://semanco-project.eu/index_html_files/SEMANCO_D3.1_20120921-1.pdf: s.n.
- Eastman, C., Teicholz, P., Sacks, R. & Liston, K., 2011. *BIM Handbook; a guide to building information modeling for owners, managers, designers, engineers, and contractors*. New Jersey: John Wiley & Sons, Inc..
- eeSemantics, 2012. *Login required*. [Online] Available at: <https://webgate.ec.europa.eu/fpfis/wikis/display/eeSemantics/Home>
- Euzenat, J. & Shvaiko, P., 2007. *Ontology matching*. Heidelberg: Springer.
- Foundation for Intelligent Physical Agents, 2002. *FIPA Device Ontology Specification*. [Online] Available at: <http://www.fipa.org/specs/fipa00091/SI00091E.html> [Zugriff am 10 April 2014].
- Fowler, M., 2010. *Domain-specific languages*. s.l.:Pearson Education.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1994. *Design patterns: elements of reusable object-oriented software*. s.l.:Pearson Education.
- gbXML, 2013. *gbXML specification schema Version 5.10*. [Online] Available at: <http://www.gbxml.org/currentschema.php> [Zugriff am 14 Februar 2014].
- Georgantas, N., 2006. *D3.2 Amigo Middleware Core: Prototype Implementation & Documentation. Amigo, IST-004182*. [Online] Available at: <http://www.hitech-projects.com/euprojects/amigo/deliverables/amigo-d3.2-final.pdf>
- Hesmos Project, 2012. [Online] Available at: <http://hesmos.eu/>
- International Electrotechnical Commission, 2008. *62541 - 1 Ed. 1.0: OPC Unified Architecture Specification - Part 1: Overview and Concepts*. s.l.:IEC Std..
- IoT.est, 2013. *Internet of Things Environment for Service Creation and Testing (IoT.est)*. [Online] Available at: <http://ict-iotest.eu/iotest/>
- Karsai, G. et al., 2009. Design guidelines for domain specific languages. *The 9th OOPSLA workshop on domain-specific modeling*, pp. 7-13.
- Krahn, H., Rumpe, B. & Völkel, S., 2010. MontiCore: a Framework for Compositional Development of Domain Specific Languages. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(5), pp. 353-372.
- Le Phuoc, D. et al., 2011. The Real World – The DERI Testbed. *IEEE Sensors*.
- Liebich, T., Stuhlmacher, K., Katranuschkov, P. & Guruz, R., 2011. *D2.1 BIM Enhancement Specification. Platform for Holistic Energy Efficiency Simulation and Lifecycle Management Of Public Use Facilitie (HESMOS) - Integrated Virtual Lab, Project ICT 26088*, http://hesmos.eu/downloads/20110831_hesmos_wp02_d21_final.pdf: s.n.
- Look, M. & Greifenberg, T., 2013. *D1.2 Report detailing Neighbourhood Information Model Semantics. COOPERATE Control and Optimisation for energy positive Neighbourhoods*, http://www.cooperate-fp7.eu/files/cooperate/downloads/COOPERATE_D12.pdf: s.n.
- Look, M. et al., 2012. Black-box Integration of Heterogeneous Modeling Languages for Cyber-Physical Systems. *GEMOC workshop*.
- OMG, 2011. *UML 2.4.1 Infrastructure*. [Online] Available at: <http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/>
- Open Geospatial Consortium, 2007. *CityGML specification document Version 0.4.0*. [Online] Available at: <http://www.citygml.org/index.php?id=1522> [Zugriff am 17 April 2014].
- OpenIoT, 2013. *Open Source Solution for the Internet of Things into the Cloud*. [Online] Available at: <http://www.openiot.eu/>
- Pesch, D., Ellis, K., Kouramas, K. & Assef, Y., 2013. *D1.1 Report on Requirements and Use Cases Specification. COOPERATE Control and Optimisation for energy positive Neighbourhoods*, http://www.cooperate-fp7.eu/files/cooperate/downloads/COOPERATE_D11.pdf: s.n.
- Probe-IT, 2013. *Pursuing ROadmaps and BEnchmarks for the Internet of Things*. [Online] Available at: <http://www.probe-it.eu/>
- Rumpe, B., 2011. *Modellierung mit UML: Sprache, Konzepte und Methodik*. s.l.:Springer.
- Rumpe, B., 2012. *Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring*. s.l.:Springer.
- Schindler, M., 2012. *Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P*. s.l.:Shaker.
- Semanco project, 2012. [Online] Available at: <http://semanco-project.eu/>
- Tzovaras, D. et al., 2012. *D1.3 State-of-Art and Industry Analysis Report. Adapt4EE Occupant Aware, Intelligent and Adaptive Enterprises*, <http://www.adapt4ee.eu/adapt4ee/files/document/deliverables/Adapt4EE-Deliverable-D1.3.pdf>: s.n.
- Völkel, S., 2011. *Kompositionale Entwicklung domänenspezifischer Sprachen*. s.l.:Shaker.