

Model Centered Architecture

Heinrich C. Mayr^[0000-0001-5770-8091], Judith Michael^[0000-0002-4999-2544],
Suneth Ranasinghe^[0000-0002-6716-7206], Vladimir A.
Shekhovtsov^[0000-0003-0227-5000], Claudia Steinberger^[0000-0002-5111-2286]

Alpen-Adria-Universität Klagenfurt,
Universitätsstraße 65-67, 9020 Klagenfurt am Wörthersee, Austria
{heinrich.mayr, judith.michael, suneth.ranasinghe,
volodymyr.shekhovtsov, claudia.steinberger}@aau.at

Abstract. This paper advocates a rigorous model focused paradigm of information system development and use. We introduce the concept of “Model Centered Architecture” that sees an information system to be a compound of various networked models, each of which is formed with the means of a Domain Specific Modeling Language. These languages are tailored to the particular circumstances of the respective system aspect. I.e., from a MOF perspective, MCA focuses on the MOF levels M_2 (definitions of the DSMLs to be used for the specification of the system and its contexts), M_1 (Specification of all System and Data Components using the DSMLs) and M_0 (the instances, i.e. models of concrete objects, functions and processes). The transformation of M_0 citizens to the respective implementation concepts (Structure \rightarrow Data, Function \rightarrow Program, Process \rightarrow Workflow) is delegated to mapping functions defined on M_2 , restricted on M_1 to the particular schemata (in the sense of mappings between the respective sets of schema instances), and instantiated on M_0 for the concrete instances. The paper shows how such model centered approach may be applied in practice using two real development projects as running examples.

Keywords: Conceptual Modeling, Model Centered Architecture, Meta Object Framework, Model Mapping, Model Transformation

1 Introduction

Motivation. Models are the fundamental human instruments for managing complexity and understanding. As such they play a key role in any scientific and engineering discipline as well as in everyday life. Many modeling paradigms evolved over time in the various disciplines leading to a huge variety of modeling languages, methods and tools that came and went. This in particular is true for Informatics, which is a modeling discipline per se, and since long tries to systematize the realm of modeling by (1) clarifying the hierarchy of model layers like e.g. in MOF (meta object framework) [26], (2) introducing ontological commitments into model hierarchies for a better semantical grounding, (3)

harmonizing various modeling approaches to unified/universal ones, and (4) providing a framework for a systematic domain specific modeling method (DSMM) ([9, 23]) design where universal approaches fail.

Since the seventies of the last century, related research and practice focuses on Conceptual Modeling. This approach basically uses a formal language the terms of which have an associated semantic interpretation (e.g. by grounding in an ontology) and a more or less transparent graphical or textual representation (supporting an efficient linguistic perception [8]). Usually, such language is embedded in a Model-/Meta-model-Hierarchy. The dimensions of conceptual modeling languages are structure, dynamics (behavior) and functionality; for instance, the Entity Relationship Model family focuses on structure, the Business Process Modeling Notation (BPMN) on dynamics and the Unified Modeling Language (UML) on all three dimensions.

A vast wealth of research has been published about conceptual modeling languages, tools and methodologies, many of them having fallen into oblivion again. Antoni Olive's fundamental contributions, however, are still present. We dedicate this paper to Antoni in deep gratitude for his inspiring work.

Related Work. We start from the observation that despite of all efforts there is still no comprehensive and consistent use of conceptual modeling in practice. Often, conceptual models are used merely as prescriptive documents, which - e.g. in the realm of software development or business process management - seldom are synchronized with the developed artefact so that reality and model are stepwise diverging.

There is a huge body of knowledge regarding Conceptual Modeling in general, and modeling paradigms and methods used for system development in particular. Thus it is not possible to give a comprehensive overview here. Also we will not discuss our own related previous research.

Model Driven Architecture (MDA) [16] and Model Driven Software Development (MDSO) [6, 19] try to master this challenge by backing model transformation [Li11] from the (conceptual) requirements model, which is defined by means of a meta-model, down to the implemented code (which clearly again is a kind of model), see also [7]. MDA can be viewed as an instance of Model Driven Development (MDD) by using the Object Management Group (OMG) standards as the core standards; i.e. Unified Modeling Language (UML), Meta Object Facility (MOF), XML Metadata Interchange (XMI), and the Common Warehouse Meta-model (CWM).

However, just as UML is not the only object-oriented modeling language, so also MDA is not the only model driven approach. There are numerous non-MDA initiatives that continue to advance the state of the art in MDD, e.g. metaprogramming [32], domain specific modeling [14], generative programming [5]. However, as there are still obstacles to overcome, e.g. regarding a bidirectional model transformation (in particular, bottom up, which would enhance synchronization), model completeness on all levels, and easy model checking,

also these approaches did not yet have an unlimited breakthrough into the developer's minds.

Models@runtime [3, 4] aims at using models (in a general sense, not specifically conceptual models) as artifacts at runtime; this is related to the wider promise that the boundary between development time and runtime artifacts should eventually disappear [2]. Run-time models are intended to enable the adaptation of a system and its context at run-time by maintaining semantic relationships between the run-time models and the running systems. This allows for analyzing and planning adaptations on the model-level (see [10] etc).

The Paper's Aim. What we will propose and illustrate within this paper is, in some ways, an add-on to the MDA/MDSD and the models@runtime methodology that is intended to attract more attention to conceptual modeling in information system development processes.

The main idea is to understand such processes as mere modeling processes (diverging from [7]) and thus focusing on models (and their meta-models) in any development step up to the running system. We, therefore, call this paradigm "Model Centered Architecture (MCA)". By MCA we will not introduce a new technology. We just aim at contributing perspectives on the power of conceptual modeling as has been done, e.g., by Antoni Olivè in [27].

The paper uses the results of two medium-term research endeavors, which we ran within the last years, as running examples. It is structured as follows. In section 2, we concentrate on models, their meta-models and Domain Specific Modeling Languages (DSML) as the key building blocks of knowledge intensive systems, around which any application and management software can be built. To make these building blocks comprehensive, the complete ecosystem context of a planned information system has to be covered. This is illustrated in section 3. In addition to that, sections 4 deals with all interfaces of such systems from a modeling perspective: the interfaces to the various user groups, to systems to be coupled, to data and knowledge sources etc. In section 5 we then present a first set of patterns for a Model Centered Architecture. The paper closes with some conclusions and an outlook on future research to be done.

2 The Model Centered Perspective

MCA is based on models, their related universal or domain specific meta-models (MMs together with the modeling languages defined in connection with these MMs, as well as on mechanisms for the transformation of models into other representations.

2.1 Background: Models and Metamodels

[11] define several features of models: (1) Mapping: A model stands for something else (its original), (2) Reduction: models map only those aspects of the original - and these possibly in a changed form - which are relevant for the given modeling

purpose, (3) Pragmatics: reflects the intended use of a model, i.e. prescription in the sense of a specification, description for explanations, simulation or formal evaluation for analysis purposes etc.

Usually, modeling is done in a way that is commonly perceived as “top-down”. For example, when designing a traditional relational database application, we start with the Relational Model as a meta-model. Using the associated Data Definition Language we define a model, the “database schema” which describes all possible states (sets of concrete tables) of the intended database. I.e., a particular database state is an extension of such schema, and again is a model: namely a model of those aspects of the original that were intended to be described by that database.

However, modeling can also be done “bottom-up”. For example, given several concrete states of relational database the schema of which is not explicitly available. So we could mine from that database an Entity-Relationship Model and represent this graphically. In this case, the data in the database are the originals.

The same data can be extensions of different models, and the same model may have different representations. As an example for the latter, suppose we are dealing with the data in a specific application, that are extensions of a, e.g., UML class diagram. This data can be represented in both, an ontological form (e.g. as OWL individuals) and using a graphical notation. These representations can replace each other depending on the current aim: being a representation for easier end user validation or a representation for enhanced reasoning. As they serve for different purposes, they have different properties as the way of representation.

It is thus clear that the hierarchy of model layers, which first was introduced in the context of Information Resource Dictionary Systems [18] and now is propagated as Metaobject Framework by [26], is helpful for understanding and managing the relations between these layers.

Meta-modeling frameworks like ADOxx¹ support the definition of Domain Specific Modeling Languages (DSMLs) and the creation of related modeling tools. Given these supporting facilities and the work of the Open Models Initiative², DSMLs and DSML development gain increasing attention [14]. The creation of a DSML as part of a comprehensive Domain Specific Modeling Method (DSMM) has been discussed in, e.g., [9] and [23]. [13] and, years ago, [15] argued modeling methods to consist of several components, that should be taken under consideration in design processes: (1) the modeling language the a syntax of which is described by means of a meta-model, the semantics by explanation or more formal descriptions, and the notation by a set of graphical elements; (2) the modeling procedure that describes how to apply the modeling language to create resulting models as well as (3) mechanisms and algorithms that work with and on these models.

Model transformation is a key technique used in MDA, where one or multiple target models are automatically generated from one or more source models ac-

¹ <http://www.adoxx.org>

² <http://openmodels.at>

ording to a transformation definition [21]. Model merging, where several models are integrated into one resulting model, is included in this definition.

2.2 MCA: The Concept

The core concept of MCA is to take models not only as representatives of underlying originals but to use them as *the core of a system* for both the addressed application functionality and the flexible definition of the system's *interfaces* as is illustrated in Fig.1 based on the MOF hierarchy.

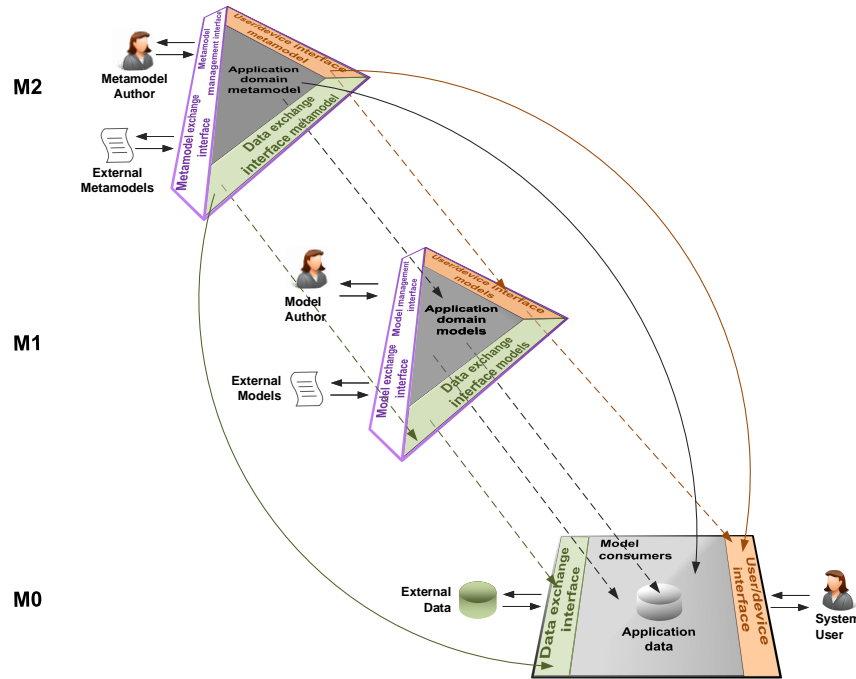


Fig. 1. Model Centered Architecture: an overview

On the M2 (meta-model) level the concepts of the DSMLs for the application domain, the user and device interfaces, and the data exchange interfaces are defined. This is done using a meta-modeling language provided on level M3 (meta-meta-model, not shown on the picture), and by specifying the symbols for language representation. The DSMLs thus are extensions of M3 and models (intensions) for M1. M2 interfaces allow for handling meta-models as MCA artifacts (*meta model management*, e.g. using authoring environments) and for integrating external meta-models (*meta-model exchange*).

On the M1 (model) level the various M2 meta-models are instantiated for a concrete application situation; the extension links are shown as dashed arrows. This leads to a (domain specific) *application model*, *user and device interface models* as well as a *data exchange model*. Again, for handling models as MCA

artifacts, management (including modeling) and exchange interfaces are defined for this level, as they are typically provided by a meta-modeling framework.

On the M0 (instance) level, the application itself results from creating extensions of the M1 application model elements (visualized in Fig.1 again by dashed arrows).

If a comprehensive DSML is defined on M2 (i.e., providing concepts for structure, dynamics and function) and used on M1, then the M0 extension form the models@runtime which are handled by an interpreter that is orchestrated by M2. The solid lines in Fig.1 visualize that correlation. Thus, the system components are implemented as model consumers and handlers which directly use and manipulate application domain and interface models to provide the necessary functionality.

MCA based approaches may work with different kinds of conceptual models as well as MMs for defining DSMLs and DSMMs. By defining them for each relevant interface and data core, it is possible to create powerful domain-specific systems. Also, model transformation is a key mechanism for MCA based approaches, since different representations, excerpts and aggregations as well as different purposes exist for models.

2.3 Running examples

To become more concrete, this paper introduces the MCA concept based on our experiences made in the HBMS³ and QuASE⁴ projects.

The *QuASE* project [30] aimed at providing an information system offering flexible means of harmonizing the stakeholders' views on communicated information (e.g. stored in industrial project repositories such as Issue Management Systems⁵ (IMS) databases) in software development projects. These means are based on terminology adaptation and the support for communication-related decisions. The core of the QuASE system is a conceptual model of the communicated information and the communication environment.

The *HBMS* project [20] aims at deriving support services from integrated models of abilities, current context and episodic knowledge that an individual had or has, but has temporarily forgotten. The core of the HBMS system is the *Human Cognitive Model (HCM)*. It preserves the episodic memory of a person in the form of conceptual models of behavior linked to context information related to these activities. The interfaces to activity recognition systems as well as multimodal user interfaces are again defined via domain specific modeling languages.

³ funded by the Klaus Tschira Stiftung gGmbH, Germany

⁴ funded by FFG (Die Österreichische Forschungsförderungsgesellschaft), Austria

⁵ e.g. Atlassian Jira

3 Models in QuASE and HBMS

3.1 Models in QuASE

Implementing QuASE as a model centered solution was motivated by the following considerations: (1) the knowledge about quality-related communications varies from company to company so that it should be separately configurable for a particular deployment site; (2) the communicated information typically is stored in project repositories (e.g. Jira databases); its conversion into knowledge (for being exploited by reasoning mechanisms approach) preferably should be integrated into the site-specific configuration.

As a consequence, a meta-model together with a visual domain-specific modeling language, the *QuASE site DSL* [29, 30], has been developed which serves for defining the *QuASE site model* as the kernel of a deployment. This model also specifies the mapping between the project repository and the modeling concepts thus allowing for an automatic generation of the knowledge base instances from the repository data (see Section 4). The DSL includes the following basic concepts: (1) *site*: owner of the given QuASE installation, e.g. a software provider; (2) *context*: units possessing certain views on communicated information e.g. projects, organizations, involved stakeholders; (3) *content*: units shaping communicated information e.g. issues/tickets; (4) *knowledge*: units encapsulating communicated knowledge.

A QuASE knowledge unit is composed of: (1) *ontological foundation*: a reference to the conceptualization of the particular piece of knowledge through ontological means; (2) *representation*: the representation of the knowledge unit in a format that could be perceived by the communicating parties (e.g. plain text); representation units are also contained in content units; (3) *resolution means*: the means of resolving understandability conflicts related to the given knowledge unit (e.g. textual explanations).

Context units possess *capabilities* to deal with knowledge units. The capabilities e.g. refer to the ability of understanding a given knowledge unit or explaining it with resolution means.

The elements of the MOF levels M0-M2 are outlined in Fig.2. Note that M3 has been omitted since a standard subset of UML-like class diagram concepts is used here which should be intuitively understandable.

The functionality of the QuASE Tool (the end user component) consists of exploiting the knowledge base by means of queries against the meta-model of the DSL, i.e. these queries refer only to M2-level concepts. This way, the resulting solution is truly model centered as it is completely customizable by defining new site models.

3.2 Models in HBMS

The HBMS MCA is backed by four meta-models: (1) The HBMS Context meta-model covering all aspects to be taken into account when it is about supporting a person, (2) the Operating Instruction meta-model that, in connection with the

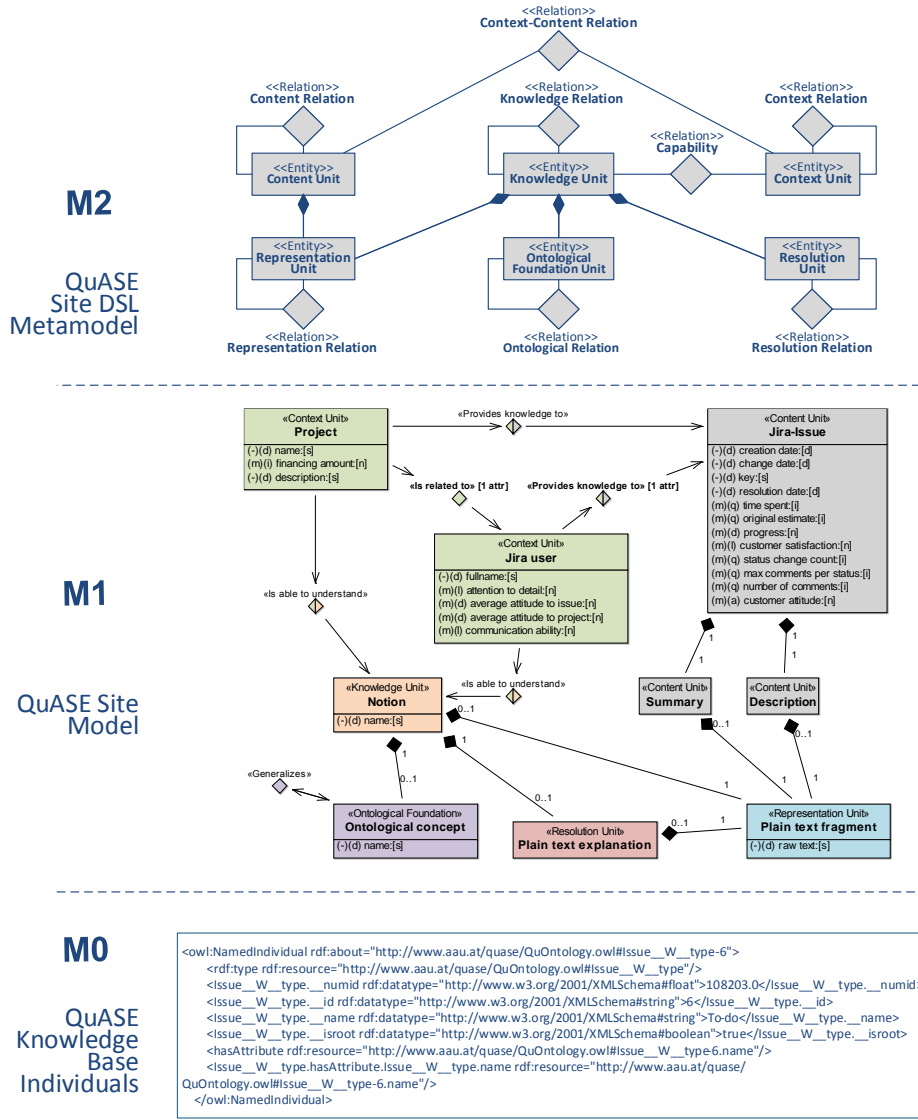


Fig. 2. QuASE MOF Levels (excerpts, simplified)

context meta-model, allows for specifying the functionality of context elements, (3) the Activity Recognition meta-model which serves for a flexible specification of the interfaces to arbitrary activity recognition systems [28], and (4) the Multimodal Support meta-model that serves for specifying the user/device interfaces for various device types. Regarding MCA as depicted in Fig.1, the Context and Operating Instruction meta-models correspond to the Application Domain meta-model, the Activity Recognition meta-model to the data-exchange meta-model, and the Multimodal Support meta-model to the user/device meta-model.

Fig.3 visualizes the HBMS MOF hierarchy, the unreadable components will be subsequently zoomed and explained. Again M3 has been omitted in the figure.

HBMS Context Model. The processing of context information gives humans the ability to adopt their behavior to the world around them [12]. As HBMS aims to actively assist individuals in activities of daily living and other situations using their own episodic knowledge, the relevant aspects of the user's context have to be known [25]. The corresponding context meta-model is structured into four clusters as shown (without details) in Fig.4:

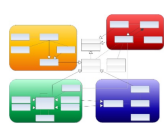



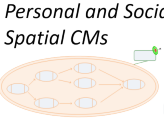
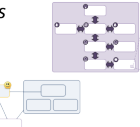




M2 Level Meta-Models	HCM-L MM 	Operating Instruction MM 	Activity Recognition MM 	Multimodal Support MM 
M1 Level Models	Behavioral CMs Environmental CMs Personal and Social CMs Spatial CMs 	insert & update of Environmental CMs 	Concrete AR Models 	Concrete Device Models 
M0 Level Data Instances	Sequence Instances Actual Context State Data Context History Data 		AR Instances <pre>Recognition { timeStamp: 25/01/17 12:39:01; Placement: action; put executing actor: Person(name: Frank, age:62); Passive-Object(name: pan, size:20cm); Passive-Object(name: pan, size:20cm); relationship: unit(pan, hot oven); }</pre>	Device Instances 

Fig. 3. MOF Levels for the HBMS-System (Meta-Models, Models and Data/Instances)

(1) The *Environmental Context* of a user: covers the resources that are utilized in operations of the assisted user or are placed as equipment in the spatial context of the user and participate in operations;

(2) The *Personal and Social Context* of a user: covers the abilities that a user holds together with the level of ability fulfilment as well as the social surrounding;

(3) The *Spatial Context* of a user: covers the location in which the user should be actively assisted;

(4) The *Behavioral Context* of a user: covers the user's relevant behavior in so-called *Behavioral Units (BUs)* that describe the possible sequences of actions

(Operations connected by Flows), their Pre- and Post-Conditions as well as their Goals.

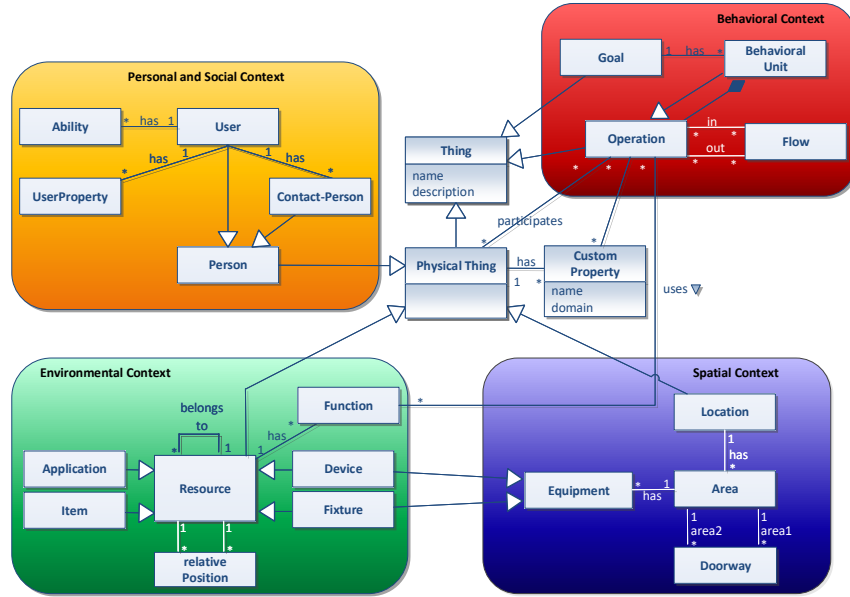


Fig. 4. HCM-L Meta-Model (excerpt)

In sum, the user’s episodic knowledge and the related context is represented and preserved at level M1 which forms the *Human Cognitive Model (HCM)*.

The HBMS context meta-model is the backbone of the lean domain specific modeling language HCM-L (Human Cognitive Modeling Language [22]). This language was designed to be as intuitively understood as possible by the relevant stakeholders in the active assistance domain [24]. It is supported by the HCM-L Modeler [1] which allows to work on the HCM.

The use of HCM is twofold: it serves (1) as a conceptual model for communication and validation purposes between stakeholders and system engineers, and (2) as a machine readable context representation allowing for retrieval, reasoning, interoperability and reuse.

4 Model Centered Interfacing

We now proceed to illustrate the concept of model-based interface design as mentioned in section 2. The targeted domains are activity recognition, multimodal support and operating instruction integration.

4.1 Model Centered Interfacing in QuASE

QuASE obtains the data from the project repositories (such as Jira databases) and converts it into knowledge stored in its knowledge base. To implement this

conversion, it includes the *knowledge base builder component* which implements a *model centered interface to project repositories*.

To support such interface, every conceptual element of a QuASE site model includes a *repository mapping specification*. This specification contains a repository query and a description of the mapping between the attributes to be returned by the query and the custom attributes of the conceptual element. It is used by the knowledge base builder as follows: during the synchronization of the knowledge base, the queries specified for the current site model are executed against the repository, the relational data returned by these queries is converted into the knowledge base individuals based on the ontological knowledge derived from the model structure, and the repository mapping specification.

The flexibility of this mapping allows large amounts of existing data to be integrated automatically. The QuASE system “can be seen as a bridge which connects end users, the data in project repositories and the (extendable) set of machine learning and natural language processing techniques” [30, p. 10] which are applicable to the data after the communication environment is described as a QuASE site model.

4.2 Model Centered Activity Recognition Interface

The implementation of an Activity Recognition interface was motivated by the fact that behavior support systems require complete knowledge about the current user behavior to provide context-aware support to its target users (e.g. elderly or disabled people in the AAL case). Moreover, in the case of HBMS we aim at supporting a person on the basis of her/his previous episodic knowledge which is to be learned via sensor based observation. Because of the limitations of current Human Activity Recognition (HAR) systems, such complete knowledge can only be established by using several HAR’s and integrating their outputs. This led us to provide, for HBMS, a model centered HAR interface that is capable of transforming heterogeneous AR data into a common representation understandable to the target system.

Fig.5 shows the main concepts of the corresponding meta-model (MOF level M2) which defines the domain specific modeling language AREM-L (Activity Recognition Environment Modeling Language):

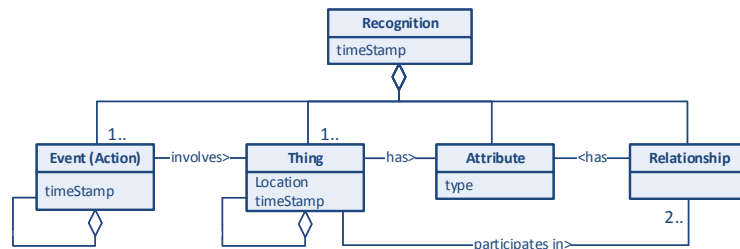


Fig. 5. Model centered HAR interface meta-model

Recognition, the top-level concept, comprises (1) a recognized *Event/Action* conceptualizing recognized simple or complex activities, (2) observed *Things*, i.e., persons or contextual objects that are connected to each other by *Relationships* and are involved in Actions, e.g., as passive elements or executing actors. For a concrete HAR to be connected, these concepts are to be instantiated appropriately on level M1. Note that then several “kinds” of events (instances of the M2 concept event), several kinds of things and relationships etc. may be defined to meet the particular HAR interface specification. The same is true for the cardinalities of the (meta-)relationships which are, for a maximum flexibility, reduced to a minimum on level M2. As an example, recognition kinds might be specified that have no event part (this is possible as there is no cardinality constraint from Recognition to Event), e.g. for covering temperature measurements of an object. On the other hand, if an event kind is specified on Level M1, it must belong to at least one Recognition kind.

At run-time, the specified interface models are used to drive the data transformation in the behavior support system.

4.3 Operating Instruction Integration

Operating instructions typically describe the core functions of a resource and give instructions for its handling. Moreover, warnings as well as typical problem situations are included. While human readers can sift through complex operating instruction and are mostly able to understand particular support information at a glance, search engines, Active Assistance Systems and other digital services need extra information to be able to use such information. Therefore, the intersection of semantic technologies and operating instructions seem to be a promising approach [31].

Fig.6 shows the refinement of the Environmental Context, describing resources and their components, functions and operating instructions relevant for HBMS-System.

Components are parts or accessories, which are necessary to prepare or to assemble the resource or which are required for special resource functions. Functions are specialized into core functions and support functions for maintenance and setup. Instructions describe how to handle functions and how to interact with the resource from the user perspective. Every instruction consists of a name and mostly an instruction text written in a certain language. Instructions that are more complex can consist of several instruction-steps, which the user is suggested to follow. Media-Objects like assembly sketches, images, audios or videos can be associated to instructions as well as to instruction steps. Warnings are operating instruction elements that are related either to an instruction directly or to the resource in general. In addition, typical problem situations can be found in an operating instruction with references to instructions to handle them.

If an operating instruction is provided online by a manufacturer in the form of structured data (e.g. using schema.org [31]), this data can be collected from the web, transformed and automatically integrated into the Environmental Context

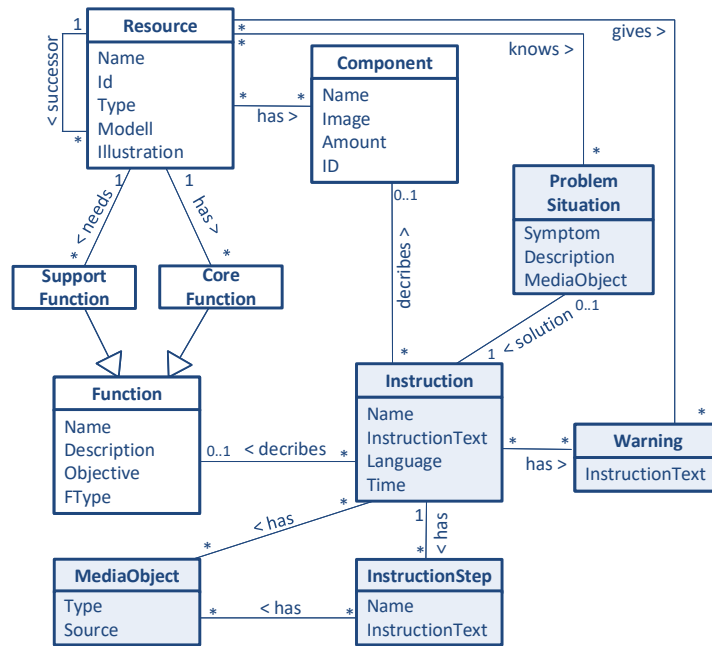


Fig. 6. Operating Instruction Meta Model as part of the Environmental Context

Model of HBMS-System using the Environmental Context meta-model for interfacing. The HCM-L Modeler then can be used for visualizing or manipulating the integrated data.

4.4 Model Centered User Interface

The user interface of a system again can be implemented based on model centered principles (as a *model centered user interface*).

Model Centered User Interface in QuASE. The QuASE system provides a model centered integration of user interface fragments into the user interface of industry issue management systems (IMS) such as Jira thus enabling Jira users to access QuASE support scenarios. This support includes (1) the concepts of the QuASE site DSL for describing the subset of model elements to be integrated into the IMS; (2) an IMS extension (e.g. Jira plug-in) which forms the control requests and transfers these to the QuASE tool; (3) the functionality of the QuASE tool for accepting IMS requests and rendering the QuASE UI fragment according to the request.

Multimodal Support Interface in HBMS. The HBMS Support Engine provides assistance to a person based on matching the person's observed actions with the current knowledge base HCM. For transmitting assistive information to

this person and for interaction purposes, a multimodal user interface is provided that works with different media types (audio, handheld, beamer, Laserpointer, light sources, etc.). Again, this interface is intended to be defined according to the MCA paradigm by introducing a domain specific modeling language; such language is currently under development in the context of a PhD work.

5 MCA: Patterns and Implementation Examples

5.1 Architectural Patterns for MCA solutions

Based on the concepts defined in the previous sections, we identify a set of architectural patterns to be implemented by the components of a MCA-based system (Fig.7).

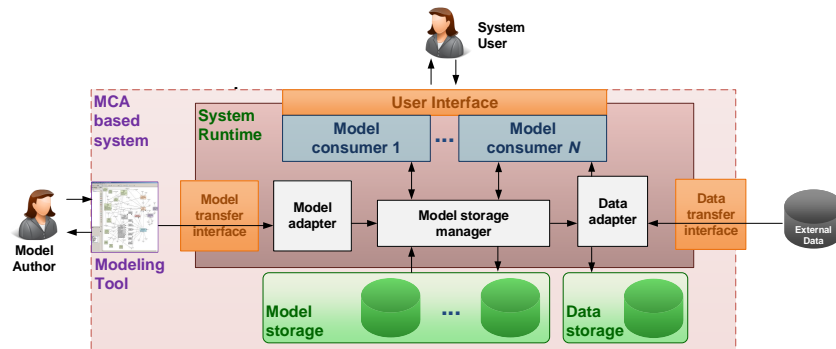


Fig. 7. Architectural patterns for MCA-based systems

1. The *modeling tool* pattern describes the means used by the model authors to create and manipulate models according to the given DSML; such tools are driven by given DSML's meta-model describing; they can be either custom built or generated using an existing meta-modeling framework.
2. The *model transfer interface* pattern describes components responsible for transferring models to runtime components.
3. The *model adapter* pattern describes components transforming the transferred models into the format understood by the rest of the system.
4. Both *model storage* and *model storage manager* patterns describe components enabling model persistence; the former describes the storage itself, the latter describes the runtime component responsible for accessing this storage.
5. The *data adapter* pattern describes those components that use models to drive the conversion of external data into the internal (system standard) representation.
6. The *model consumer* pattern describes the components which use the adapted models to provide the functionality of the MCA-based solution.

needs support. The HBMS system implements the MCA patterns as follows (see Fig.9):

1. The *modeling tool* implements the *modeling tool* pattern. It is used for creating and maintaining HCM-L and AREM-L models, and communicates with the HBMS kernel by means of the *model transfer interface* implementing the related pattern.
2. The *HCM-L-OWL converter* implements the *model adapter* pattern. It transforms HCM-L models into OWL2 representation used by the HBMS kernel.
3. The *knowledge base* implements the *model storage* pattern being a triple store holding HCM and behavioral data. The kernel communicates with it through the *Data Management Subsystem* that implements the *model storage manager* pattern.
4. The *activity recognition system (ARS) adapter* is a middleware listening to the data coming from an ARS and making it HBMS-compliant. It implements the *data adapter* pattern being driven by the AREM-L description of the particular ARS interface;
5. The *HBMS kernel* contains the following components implementing the *model consumer* pattern: (1) *Observation Engine*: responsible for communicating to ARS through the ARS adapter; (2) *Behavior Engine*: responsible for handling the behavior data arriving from the Observation Engine in context of the current HCM; (3) *Support Engine*: responsible for controlling the behavior of the assisted users through the multimodal user interface.

MCA support infrastructure. Within the context of the HBMS and QuASE projects we established a flexible software development infrastructure to back MCA-based applications. This infrastructure is subdivided into:

1. *modeling tool infrastructure*: the infrastructural elements assisting the developers of meta-models to be used for defining MCA models such as the means for selecting the subset of the models to be transferred to the runtime components;
2. *model transfer interface infrastructure*: the elements assisting the developers of the model transfer interface such as (1) the *transfer script* callable from the modeling tool (it converts the selected models into the transfer format and uploads them to the system runtime), (2) the Java implementation of the *model listener*: a component of the kernel which listens to the communication port used for uploading the models, captures the uploaded models, and makes them available to the rest of the kernel;
3. *kernel infrastructure*: the elements assisting the developers of the kernel components such as the implementation of (1) the *model mapper* transforming models into a set of internal objects to be used by the rest of the kernel, and (2) the *model serializer* transforming OWL2 model representations into a triple store.

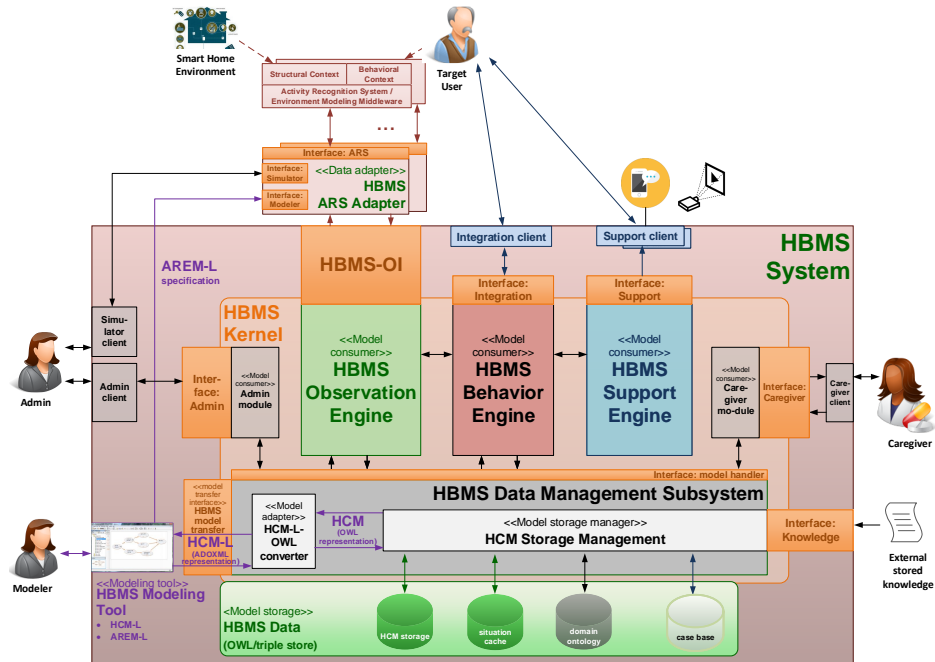


Fig. 9. HBMS system architecture as MCA implementation

Applied in the development process, this infrastructure decreases the development effort by taking the responsibility for the technical issues related to the implementation of MCA-based solutions: from defining the models to utilizing the models at runtime.

6 Outlook

With the MCA paradigm we want to contribute to a more comprehensive use of conceptual modeling in practice. The paradigm provides transparent means of synchronizing models and developed artifacts on all software development stages, and also in the running system. By applying this paradigm, the conceptual models are not restricted to being the prescriptive documents which eventually diverge from what is used by the running system, instead, they are considered as crucial system artifacts directly influencing the functioning of the system interfaces and components at runtime. In fact, they become "first-class citizens" of the running system.

The MCA paradigm allows for increasing the adaptability of software solutions by providing DSMLs as application specific and flexible means of (1) specifying the application context and the relevant interfaces comprehensively, and (2) driving the runtime behavior of the system.

The MCA paradigm could be extended in future along the following directions:

- To investigate in more detail the ingredients of interface modeling languages; possible research topics here could be the formalization of such languages, related quality characteristics etc.
- To review the traditional notions of quality for conceptual models w.r.t. their validity in the MCA realm.
- To investigate the adaptability of the MCA paradigm in agile software development.

References

1. Al Machot, F., Mayr, H.C., Michael, J.: Behavior Modeling and Reasoning for Ambient Support: HCM-L Modeler. In: Proc. of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA-AIE (2014)
2. Baresi, L., Ghezzi, C.: The Disappearing Boundary Between Development-Time and Run-Time. In: Proceedings of the FSE/SDP workshop on Future of software engineering research, pp. 17-22. ACM (2010)
3. Bencomo, N., France, R.B., Cheng, B.H., Aßmann, U.: Models@ run. time: Foundations, Applications, and Roadmaps, LNCS, Vol. 8378. Springer (2014)
4. Blair, G., Bencomo, N., France, R.: Models@ run. time. *Computer* 10, 22-27 (2009)
5. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional, Boston, Massachusetts, USA (2000)
6. Embley, D.W., Kurtz, B.D., Woodfield, S.N.: Object-oriented Systems Analysis: A Model-Driven Approach. Prentice-Hall, Englewood Cliffs, New Jersey (1992)
7. Embley, D.W., Liddle, S.W., Pastor, O.: Conceptual-Model Programming: a Manifesto. In: Handbook of Conceptual Modeling, pp. 3-16. Springer (2011)
8. von Foerster, H.: Perception of the Future and the Future of Perception. *Instructional Science* 1, 31-43 (1972)
9. Frank, U.: Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In: Reinhartz-Berger, I. et al. (eds.): Domain Engineering. pp. 133-157, Springer (2013)
10. Heinrich, R. et al.: Runtime Architecture Models for Dynamic Adaptation and Evolution of Cloud Applications. Universität Kiel (2015)
11. Hesse, W., Mayr, H.C.: Modellierung in der Softwaretechnik. eine Bestandsaufnahme. *Informatik-Spektrum*, 31, 377-393 (2008)
12. Hoareau, C., Satoh, I.: Modeling and Processing Information for Context-Aware Computing. A Survey. *New Gener. Comput.* 27(3), pp. 177-196 (2009)
13. Karagiannis, D., Kühn, H.: Metamodelling Platforms. In E-Commerce and Web Technologies, K. Bauknecht, A. M. Tjoa and G. Quirchmayr, Eds. LNCS. Springer, Berlin, Heidelberg, p. 182 (2002)
14. Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.): Domain-Specific Conceptual Modeling: Concepts, Methods and Tools. Springer (2016)
15. Kaschek, R., Mayr, H.C.: A Characterization of OOA Tools. Assessment of Software Tools, 1996., Proceedings of the Fourth International Symposium on, pp. 59-67. IEEE (1996)
16. Kleppe, A.G., Warmer, J.B., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc. (2003)

17. Lewis, P.R. et al.: Architectural Aspects of Self-Aware and Self-Expressive Computing Systems: From psychology to engineering. *Computer* 48, 62-70 (2015)
18. Leymann, F., Altenhuber, W.: Managing Business Processes as an Information Resource. *IBM systems journal* 33, 326-348 (1994)
19. Liddle, S.W.: Model-Driven Software Development. In *Handbook of Conceptual Modeling*, pp. 17-54. Springer Berlin Heidelberg (2011)
20. Mayr, H. C. et al.: HCM-L: Domain-Specific Modeling for Active and Assisted Living. In: Karagiannis, D.; Mayr, H. C.; Mylopoulos, J. (eds.): *Domain-specific conceptual modeling. Concepts, methods and tools*. pp. 527-552, Springer (2016)
21. Mens, T.; Czarnecki, K.; van Gorp, P.: A Taxonomy of Model Transformation. *Proc. Dagstuhl Seminar on "Language Engineering for Model-Driven Software Development"*. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl (2005)
22. Michael, J., Mayr, H.C.: Conceptual Modeling for Ambient Assistance. In: Ng, W., Storey, V.C., Trujillo, J. (eds.): *Conceptual Modeling - ER 2013*, pp. 403-413. Springer (2013)
23. Michael, J., Mayr, H.C.: Creating a Domain Specific Modeling Method for Ambient Assistance. In: *International Conference on Advances in ICT for Emerging Regions (ICTer2015)*. IEEE (2015)
24. Michael, J., Mayr, H.C.: Intuitive Understanding of a Modeling Language. In: *Proc. of the Australasian Computer Science Week Multi-conference (ACSW'17), Asia Pacific Conference on Conceptual Modeling (APCCM)*, pp. 1-10. ACM (2017)
25. Michael, J., Steinberger, C.: *Context Modeling for Active Assistance*, submitted for publication
26. Object Management Group OMG: *Meta Object Facility™ (MOF™) Core*, URL: <http://www.omg.org/spec/MOF/>, last accessed 09.08.2016
27. Olivé, A., Cabot, J.: A Research Agenda for Conceptual Schema-Centric Development. In: *Conceptual Modelling in Information Systems Engineering*, pp. 319-334. Springer (2007)
28. Ranasinghe, S., Al Machot, F., Mayr, H.C.: A Review on Applications of Activity Recognition Systems with Regard to Performance and Evaluation. *International Journal of Distributed Sensor Networks* 12, (2016)
29. Shekhovtsov, V.A., Mayr, H.C., Kop C.: Facilitating Effective Stakeholder Communication in Software Development Processes. In: Nurcan, S., Pimenidis, E. (eds.): *Information Systems Engineering in Complex Environments, LNBIP, Vol. 204*, pp. 116-132. Springer (2015)
30. Shekhovtsov, V.A., Mayr, H.C.: View Harmonization in Software Processes: from the Idea to QuASE. In: Mayr, H.C., Pinzger, M. (Hrsg.). *INFORMATIK 2016*, 26.-30. September 2016, Klagenfurt, Österreich. *Proceedings*, pp. 111-123, LNI, Vol. P-259, GI, (2016)
31. Steinberger, C., Michael, J.: *Semantic Mark-Up of Operating Instructions for Active Assistance*, submitted for publication
32. Štuikys, V., Damaševičius, R.: *Meta-Programming and Model-Driven Meta-Program Development: Principles, Processes and Techniques*, Springer (2012)