

# Abstraction Levels in the Abstract State Machine (ASM) Method for System Specification

Jan Kubovy<sup>1\*</sup>, Mariam Rady<sup>2</sup>, Dagmar Auer<sup>1</sup>, and Josef Küng<sup>1</sup>

<sup>1</sup> Institute for Application Oriented Knowledge Processing  
Johannes Kepler University in Linz  
jkubovy@faw.jku.at, dauer@faw.jku.at, jkueng@faw.jku.at  
<http://www.faw.jku.at>

<sup>2</sup> Christian Doppler Laboratory for Client-Centric Cloud Computing  
Institute for Application Oriented Knowledge Processing  
Johannes Kepler University in Linz  
m.rady@cdcc.faw.jku.at  
<http://www.faw.jku.at>

**Abstract.** The ASM method allows describing a system formally on a certain abstraction level [see 1–3; 8]. However this abstraction level can change during the process of designing a system as a result of refinements. The way a system needs to be described on an abstract level usually differs from the way a system is described when more concrete specifics are known. In this paper we introduce two basic abstraction levels, their semantics, show how to realize transition from one abstraction level to another and how to preserve consistency in both levels in case of changes.

**Keywords:** ASM, formal methods, abstraction level, refinements, transition, consistency

## 1 Introduction

The ASM Method allows to describe a system formally on a certain abstraction level [see 1–3; 8]. It supports the system development process starting with the problem specification through several steps till implementation by incrementally refining the system model.

Starting with an abstract model of the system, it is continuously refined within the ASM method, i.e. enriched and thus made more specific. The typical tool for developing models with the ASM method is a text editor. All models are described in documents, consisting of formal descriptions and natural language explanations. Typical document types are project proposals, system specifications or design documents. They all refer to system under design but

---

\* This work was supported in part by the Austrian Science Fund (FWF) under grant no. TRP 223-N23.

have different purpose and thus differ in the abstraction level and the target audience. When the system needs to be changed, it is very hard to keep the models consistent over the abstraction levels, described in several documents using specific conventions designed for different target audiences. Thus some kind of relation or linking between corresponding modeling elements on the different abstraction levels is needed.

We illustrate the problem by an example in software development, which is a typical domain to deploy the ASM method. Two main levels of abstraction are defined in this context, the business level and the technical level. The first focuses on the needs of business people while technical people are the target audience for the latter. We do not deal with the differences between formal and informal methods in this paper. We assume that all models on all abstraction levels are specified using the ASM method. The ideas presented in this paper are not limited to software development and can be adapted to other domains.

Section 2 shows how to systematically specify a system on an abstract level (namely the business level), to provide a readable natural language description for the target audience and be prepared for consistent vertical refinements as well. Most of the model refinement steps are part of the technical abstraction level which is described in Section 3 with special emphasis on the structure and description elements used for specifying functions and rules. The developed *4-level transition approach* is described in Section 4. Section 5 shows how to assure consistency in the case of changes in one of the abstraction levels. Commenting the results including an outlook on further research closes our discussion of the topic in section 6.

### 1.1 Naming conventions

The names of the ASM constructs used in this document will follow following conventions:

**universe names** are written in plural in uppercase letters using an underscore (“\_”) to separate words.

**function names** are written in camel case with first letter in lowercase.

**rule names** are written in camel case with first letter in uppercase.

**location and parameter names** are written in singular (or plural for sets only) in camel case with first letter in lowercase.

We assume that a camel case singular version of a universe is a member of such a universe, as shown in Equation 1.

$$connectingObject \in CONNECTING.OBJECTS \quad (1)$$

## 2 ASM on business level of abstraction

The specification of a system starts on the business level. Let’s assume that the intended documents of a system are: project proposal, project specification

and technical specification. Then the first two, the project proposal and the project specification fit the *business level of abstraction* described in this paper. The assumed audience for this type of documents are *technically aware business people*.

The development process of a system usually starts with a project proposal done by the *product owner*. This project proposal is a starting point for the *software analyst* who will capture the requirements and write a document representing, as said in [4]:

“[...] a succinct process-oriented model of the to-be-implemented piece of *real world*, transparent for both the customer and the software designer so that they can serve as the basis for the software contract, a document which binds the two parties involved.”

The step between the project proposal and the analysis document may be considered as one of many refinement steps. Those are the main common documents written on the first level of abstraction, which we identify as business level.

## 2.1 Business level document structure

A formal specification on business level includes a significant amount of abstract rules and abstract derived functions. The amount of those abstract functions and rules will decrease during refinement steps, even on this level of abstraction. However the goal is not a complete elimination of those abstract rules or abstract derived functions. Also significant amount of other concrete details, including data types, may be kept abstract on this level of abstraction. The resulting documents usually tend to be natural-language documents rather than a collection of pseudo-code. The formal core of the description should be captured in the ASM description, still additional informal description occurs frequently in such documents. An example of such description can be seen in [4; 5], where the Business Process Model and Notation (BPMN) 2.0 Specification [9] is described formally. We will use and build on those examples in this paper.

## 2.2 Using formal elements in an informal text

The document may look like rather a continuous text with formal elements embedded. Blocks of pseudo-code are present but rather small and using a significant number of abstract derived functions and abstract rules. Consider the example in [5] and assume we need to get the information, which process instance a concrete token [6] is in. For this purpose we may introduce an `instanceOfToken(t)` function which will provide us with the required information. We may want to get the same information for running processes or instantiated activities. For this purpose we introduce, in the same way, two more functions: `instanceOfProcess(p)` and `instanceOfActivity(a)`. In those proposed conventions all three parameters, `t`, `p` and `a` are arbitrary structures with

no defined type yet. This way we can speak about instances of tokens, processes or activities in an informal text with minimal impact on the fluency and naturalness of the text.

In order to introduce such a formal element without a type those elements have to be explained. One possibility is to introduce some conventions or guidelines. In our example we introduced a “*propertyOfSubject*” naming convention for functions to preserve the naturalness of the text. A similar “*ActionOfSubject*” naming convention can be introduced for rules. This will lead to renaming of rules shown in [4; 5] as shown in Equation 2 and Equation 3.

$$\text{WorkflowTransition} \Rightarrow \text{TransitionOfWorkflow} \quad (2)$$

$$\text{GatewayTransition} \Rightarrow \text{TransitionOfGateway} \quad (3)$$

The intention is to introduce semantics into the naming conventions to be able to use formal elements in natural texts while maintaining the option to keep most of the properties abstract, including types, especially in the beginning of the modeling process. Those conventions enable the transition described in Section 4 and also help to preserve the consistency between the abstraction levels described in Section 5. The concrete conventions are flexible and up to the author of the concrete system to be designed. The examples in this paper are only one possible option.

Similar way we would write a multi-parameter function or rule. For example a function name which represents all tokens in a sequence flow (arc) and a certain process instance, using the introduced conventions may be seen in Equation 4.

$$\text{tokensInArcAndInstance}(a, i) \quad (4)$$

### 2.3 Adding types

One of the first refinement steps is to start using types, i.e. universes. This may be the difference between a project proposal and the project specification. Consider the function mentioned in the previous section. Now it may make sense to say that each of them is meant for a different data type by introducing their *declaration signatures*, e.g.: `instanceOfToken : TOKENS → INSTANCES` for getting the instance of a token, `instanceOfProcess : PROCESSES → INSTANCES` for getting the instance of a process and `instanceOfActivity : ACTIVITIES → INSTANCES` for getting the instance of an activity. Using the ASM method two signatures are possible:

**declaration signature** used for declaring rules or functions and their parameter universes and the return value universe for functions.

**definition signature** used for defining, calling or referring to a rule or function.

Rules and derived functions may also define their bodies after this type of signature.

Where the parameter universes are not yet defined, they may be discovered using the naming conventions mentioned in Section 1.1 and Section 2.2. This

allows using formal elements in natural text as described in Section 2.2 without breaking the fluency of the text.

The model can be refined by adding more formal information to the description. The implicit semantics within the names of the mentioned functions are now explicitly expressed. The properties of these types are still abstract and so are relations between them. After several refinement steps some of these relations or properties may appear obvious, but their explicit formal definition is intended to be done in the technical level.

### 3 ASM on technical level of abstraction

Ordinarily, after the product owner and the software designer agree on the model, the software designer proceeds with a *technical specification* of the intended system. Such a specification can pass through several refinements steps before the first implementation attempt. Also after the first release is deployed additional refinement steps may occur as change requests [11]. The direction is to already make more product specific decisions and go into more detail. Yet still concrete implementation decisions do not have to be made. The model can still be kept abstract enough to keep the implementation language and similar questions open. But the emphasis should be already put into technical, abstract-as-possible and complete-as-necessary, direction.

The specification already went through several refinement iterations before being transformed to this level of abstraction. The project proposal or similar document from the business level is the basis for the technical specification. But there is no formal link between those documents on the different abstraction levels, which would preserve the consistency of both of them. The resulting two documents are typically kept concurrent till the first implementation release, accepted by the *product owner*, is deployed. Later the two documents detach from each other. The technical specification may live further and be refined as a consequence of Request for Change (RFC) while the product specification becomes obsolete.

#### 3.1 Technical level document structure

The emphasis passes from the natural text document described in Section 2 to rather pseudo-code constructs. The complexity of the pseudo-code constructs increases and the amount of abstract functions and rules decreases. Yet still not all have to be eliminated. Types in this abstraction level may be mostly defined. So may be their properties and relations. The usage of types may lead to refinement of names of some functions or rules. Assume the functions mentioned in Section 2.3. The consequence of abstraction level change from business level to technical level may lead to cutting of parts of the function or rule names. Based on the proposed naming convention refinement those name parts are superfluous since the *property* and *subject* are already formally defined using types. E.g.:

– `instanceOfToken` : TOKENS  $\rightarrow$  INSTANCES

- `instanceOfProcess` : PROCESSES  $\rightarrow$  INSTANCES
- `instanceOfActivity` : ACTIVITIES  $\rightarrow$  INSTANCES

may lead to:

- `instanceOf` : TOKENS  $\rightarrow$  INSTANCES
- `instanceOf` : PROCESSES  $\rightarrow$  INSTANCES
- `instanceOf` : ACTIVITIES  $\rightarrow$  INSTANCES

With this refinement, function and rule overloading is enabled. To still use this functions in natural language text, we have to be more concerned about the parameter naming conventions than we were in the business level. E.g., `instanceOf(t)` may lead to confusion about the parameter type. Thus the usage of `instanceOf(token)` in natural language text is recommended. Also the relations between the different types can be defined, which introduces a structure into types[4], e.g., as shown in Equation 5.

$$\text{NODES} = \text{ACTIVITIES} \cup \text{EVENTS} \cup \text{GATEWAYS} \quad (5)$$

### 3.2 Selectors and properties of universes

#### Listing 1. Selector notation

```
TOKENS.instance  $\rightarrow$  INSTANCES
PROCESSES.instance  $\rightarrow$  INSTANCES
ACTIVITIES.instance  $\rightarrow$  INSTANCES
```

An analogy between function application and selection, described in [10] and shown in Equation 6, may be used to introduce namespaces and a way how to define properties of universes. Using this analogy we may refine the functions: `instanceOf` : TOKENS  $\rightarrow$  INSTANCES, `instanceOf` : PROCESSES  $\rightarrow$  INSTANCES and `instanceOf` : ACTIVITIES  $\rightarrow$  INSTANCES and write them as shown in Listing 1.

$$f(x) \equiv x.f \quad (6)$$

Obviously this is less natural language text oriented approach than the versions of those functions in Section 2. However the link between all the versions is preserved. Furthermore, we now can use universes as namespaces to functions as shown in Listing 2.

#### Listing 2. Namespaces

```
TOKENS {
  instance  $\rightarrow$  INSTANCES
  arc  $\rightarrow$  CONNECTING_OBJECTS
}
```

This is just another way how to write the previous definition of a function, but introduces a way to define properties of universes and to aggregate functions into logical units. We are simply defining a property **instance** and **arc** for any data structure from the universe **TOKENS**. Additionally the name of the functions is refined by cutting of the “Of” ending from the function names, as it lost its semantical meaning when the parameter was transformed to a selector. Those refinements allow to create more technical-oriented descriptions decreasing the ambiguity, since the informal parts will be reduced.

In case of multi-parameter function the first parameter may be used as selector and the rest as parameters of the function as shown in Equation 7.

$$f(x, y, z) \equiv x.f(y, z) \quad (7)$$

The function **tokensInArcAndInstance(a, i)** introduced in Section 2.2 will be reformatted as shown in Equation 8.

$$\text{CONNECTING\_OBJECTS.tokens} : \text{INSTANCES} \rightarrow \text{SET(TOKENS)} \quad (8)$$

Also note that at this level of abstraction both notations, with or without selectors, are considered equal. Both of them can be used at the same time in the technical level document.

#### 4 Transition between the abstraction levels

The transition between the abstraction levels from business level to technical level can be (semi-)automatic. The idea of such a transition is to introduce *transition rules* which will enable extracting the formal parts of the document at the business level and generate the skeleton for the technical level. A manual interaction may be necessary depending on the abstraction degree of the business level. The ability of (semi-)automatic transition heavily depends on the conventions used on the business level discussed in Section 2 and Section 3.

Transition between abstraction levels can be regarded as a specific type of refinement, where one function is replaced by another. The transition between the abstraction levels in this paper can be seen as a way of a syntax refinement. For example the refinement where types are added to the specification can be seen in Equation 9 and Equation 10.

$$\begin{aligned} \text{instanceOfToken}(t) &\Longrightarrow_{T_1} \\ &\Longrightarrow_{T_1} \text{instanceOfToken} : \text{TOKENS} \rightarrow \text{INSTANCES} \end{aligned} \quad (9)$$

$$\begin{aligned} \text{arcOfToken}(t) &\Longrightarrow_{T_1} \\ &\Longrightarrow_{T_1} \text{arcOfToken} : \text{TOKENS} \rightarrow \text{CONNECTING\_OBJECTS} \end{aligned} \quad (10)$$

The naming refinement is shown in Equation 11 and Equation 12.

$$\begin{aligned} \text{instanceOfToken} &: \text{TOKENS} \rightarrow \text{INSTANCES} \Longrightarrow_{T_2} \\ \Longrightarrow_{T_2} \text{instanceOf} &: \text{TOKENS} \rightarrow \text{INSTANCES} \end{aligned} \quad (11)$$

$$\begin{aligned} \text{arcOfToken} &: \text{TOKENS} \rightarrow \text{CONNECTING\_OBJECTS} \Longrightarrow_{T_2} \\ \Longrightarrow_{T_2} \text{arcOf} &: \text{TOKENS} \rightarrow \text{CONNECTING\_OBJECTS} \end{aligned} \quad (12)$$

The transition between function application and selection, if types of parameters are already defined. Basically we collect all the functions and group them by the type of the first parameter and then do the transition shown in Equation 13 and Equation 14.

$$\begin{aligned} \text{instanceOf} &: \text{TOKENS} \rightarrow \text{INSTANCES} \Longrightarrow_{T_3} \\ \Longrightarrow_{T_3} \text{TOKENS.instance} &\rightarrow \text{INSTANCES} \end{aligned} \quad (13)$$

$$\begin{aligned} \text{arcOf} &: \text{TOKENS} \rightarrow \text{CONNECTING\_OBJECTS} \Longrightarrow_{T_3} \\ \Longrightarrow_{T_3} \text{TOKENS.arc} &\rightarrow \text{CONNECTING\_OBJECTS} \end{aligned} \quad (14)$$

The step of grouping such function signatures ( $T_4$ ) will then result as shown in Listing 3.

**Listing 3.** Namespace grouping

```
TOKENS {
  instance → INSTANCES
  arc → CONNECTING_OBJECTS
}
```

This example illustrates the *4-step abstraction level transition approach* ( $T_{1,2,3,4}$ ) proposed in this paper. Neither the number of steps nor their definition should be considered complete. Both, the number of steps and their definition may be bent to the specific needs. Nor should the placement of a concrete refinement step in an abstraction level class, business level or technical level, be considered as fixed. In this paper we expect the 1<sup>st</sup> step ( $T_1$ ) to be in done on the business level and the 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> ( $T_{2,3,4}$ ) steps on the technical level.

The intended (semi-)automatic transition between the two defined levels of abstraction will apply those *transition rules* to existing formal objects in the business level documentation. Based on their names the types of parameters and return values can be detected. We assume that the function and rule names in the business level document are based on the naming conventions discussed in Section 2.2.



## 5 Preserving consistency between the abstraction levels

Preserving consistency will pay itself off as soon as the *product owner* will apply RFC to the original proposal. Usually such RFC [11] is submitted as a distinct document. Using the conventions discussed in this paper the product owner will be able to make changes to the original proposal and the technical level specification skeleton may be (semi-)automatically updated.

The idea of preserving the consistency between the business level and technical level relies on the transition rules discussed in Section 4. After an update in the business level, the technical level document skeleton has to be updated using the mentioned transition rules. New formal objects from the business level will be added to those in the technical level (semi-)automatically. Furthermore, formal objects which are no longer available in the business level are marked, not deleted, in the technical level for further handling. The product designer will see all the differences and changes, made by the product owner, in the technical level specification and will refine the changes towards the implementation. Consistency test should also be possible vice-versa, starting at the technical level up to the business level.

## 6 Conclusion

Based on the ASM method, abstraction levels are introduced to support the model refinement during the software development process. The business level with its refinement steps is intended to be the communication basis between the product owner and the product designer. Similarly for the technical level, which is dedicated to the product designer and the product implementer.

The ASM models are described in text documents, as no adequate tool support is available for now. Thus keeping the models consistent throughout the refinement process or when handling change requests later on, is challenging. Being able to unambiguously and clearly pass the information of what is going-to-be-implemented in order to match the requirements is often not trivial.

We proposed conventions for describing functions, rules and associated concepts as well as a stepwise refinement process based on transition rules. The *4-step abstraction level transition approach*, illustrated in this paper, shows one possible way how to deal with refinements. The conventions and rules are also used vice versa as the basis for consistency checks, which are needed to guarantee that technical level models are consistent with business level ones.

For now this approach has only been tested on some small examples [7] but expect to be promising. It will soon be tested on our largest ongoing project based on the ASM method. The indented transition rule framework should make the transition machine configurable in such a way that it can be used for different system designs by adjusting the rules and conventions to satisfy the concrete needs. The results will be used to develop further ideas concerning tool support for working with the ASM model for system specification.

## References

- [1] Börger E, Schulte W (1999) Modular design for the java virtual machine architecture
- [2] Börger E, Schulte W (1999) A programmer friendly modular definition of the semantics of java. In: Formal Syntax and Semantics of Java, Springer-Verlag, London, UK, UK, pp 353–404
- [3] Börger E, Stärk RF (2003) Abstract State Machines - A Method for High-Level System Design and Analysis. Springer-Verlag
- [4] Börger E, Thalheim B (2008) A method for verifiable and validatable business process modeling. *Advances in Software Engineering*, LNCS 5316:59–115
- [5] Börger E, Thalheim B (2008) Modeling workflows, interaction patterns, web services and business processes: The asm-based approach. In: *Proceedings of the 1st international conference on Abstract State Machines*, B and Z, Springer Berlin / Heidelberg, pp 24–38
- [6] van Gorp P, Dijkman R (2012) A visual token-based formalization of bpmn 2.0 based on in-place transformations. *Information and Software Technology* 2(55):365–394
- [7] Kubový J, Rady M, Auer D, Küng J (2013) Transition between different abstraction levels in an abstract state machine (ASM) ground model. 2013 24rd International Workshop on Database and Expert Systems Applications pp 227–230
- [8] Kutter PW, Pierantonio A (1997) The formal specification of oberon. *Journal of Universal Computer Science* 3(5):443–503
- [9] Object Management Group (OMG) (2011) Business process model and notation (BPMN) 2.0. [www.omg.org/spec/BPMN/2.0](http://www.omg.org/spec/BPMN/2.0)
- [10] Odersky M (2000) Objects + views = components? In: *Abstract State Machines 2000*, Springer-Verlag, Lecture Notes in Computer Science
- [11] Office of Government Commerce (OGC) (2007) ITIL Version 3: Service Operation. The Stationery Office