

Investigations of a Method for Comparing Process Models

B. Andersson

June, 2001

Master's thesis¹

Department of Computer and Systems Sciences
Stockholm University / Royal Institute of Technology

Abstract

Sometimes there is a need to compare process models constructed in different process modeling languages to each other to determine if they model the same phenomena. Sometimes there is a need to translate a process model made in one modeling language to another language. A basic demand on the translated model is that it models the same phenomena as the original model. In both cases there is a need to determine if they have similar semantics. How should one go about when trying to compare models made in different process modeling languages for semantic similarity?

A method for doing model comparisons is proposed and validated in this thesis. The basic idea in the method is to construct a reference model, founded on some general concepts, that is used as a norm when doing the comparisons. The notational constructs in the models are then mapped onto the concepts in the reference model and a comparison can be made based on the conceptual content in the models. The method is validated by using it in a modeling exercise. The result of the validation is that the method can be used to determine whether two process models are similar or not.

¹This thesis corresponds to 20 weeks of full-time work

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem	1
1.3	Purpose and goals	2
1.4	Method motivation	2
1.5	Method	3
1.6	Investigation approach	4
1.7	Intended audience	6
1.8	Thesis scope	6
2	Theoretical Framework	6
2.1	Ontology	6
2.1.1	Introduction	6
2.1.2	Thing, Property, Attribute, State, Law	7
2.1.3	Event, Transformation, Behaviour, History, Acts on	7
2.1.4	System, Subsystem, Composition, Structure	8
2.1.5	Comments	9
2.2	A process metamodel	9
2.3	Definition of process similarity	11
3	Process Modeling Languages	12
3.1	Introduction	12
3.2	BPM	13
3.2.1	Concepts	13
3.3	SDL	14
3.3.1	Concepts	14
4	Modeling exercise	16
4.1	The coffee machine	16
4.2	Models	17
4.3	A mapping schema	17
4.4	Comparison and translation	20
4.4.1	Comparison	20
4.4.2	Result of comparison	22
4.4.3	Translation	22
4.4.4	Result of translation	25
5	Conclusion	25
A	BPM models	i
B	SDL models	iii

1 Introduction

1.1 Background

There are processes everywhere. Opening an account in a bank is a process, making a cup of coffee is a process, assembling a car is a process. The following quote points out there are several ways to describe real and abstract phenomena, where a process description is one of the ways.

“A circle is a locus of all points equidistant from a given point.”
“To construct a circle, rotate a compass with one arm fixed until the other arm has returned to its starting point.” It is implicit in Euclid that if you carry out the process described in the second sentence, you will produce an object that satisfies the definition in the first. The first sentence is a state description of a circle; the second a process description.

These two modes of apprehending structures are the warp and weft of our experience. Pictures, blueprints, most diagrams, and chemical structural formulas are state descriptions. Recipes, differential equations, and equations for chemical reactions are process descriptions. The former characterise the world as sensed; they provide the criteria for identifying objects. The latter characterise the world as acted upon; they provide the means for producing or generating objects having the desired character”[1]

Any description is a model, and the model can be expressed using the symbols and rules of a language. To model a process, a process modeling language is used. There are several different process modeling languages in existence, all capable of describing such processes as were exemplified above. Some of these languages are standardised and are widely used, like Activity Diagrams in UML[2]. Some are very informal and used only locally, perhaps in a small firm. Process models are often expressed using some diagrammatical notation and in many cases there is computer support for generating the diagrams.

1.2 Problem

Sometimes there is a need to translate process models constructed in one language to another. For example, a company that has made and accumulated a huge number of models over the years, realises that the administrative overhead in keeping these models constructed in different languages are too large. The company decides to standardise. The work done, however, should not and can not be thrown away and this creates a need to translate from the old languages to the new one.

Sometimes there is a need to compare process models made in different languages to determine if they model a similar thing. For example, two

companies that are using different modeling standards has modeled some process that is common to both companies. No company can afford to adopt or use the other ones standard, but it is essential that they both can agree on the fact that they are modeling the same process. So here arises the need for being able to compare the models.

1.3 Purpose and goals

The purpose of this thesis is to propose and investigate a method for doing a comparison of two process models to determine if they are semantically similar. The goals are to describe the method and to do a method validation. The validation is made by trying the method in a modeling example.

1.4 Method motivation

How should one go about when trying to compare models made in different modeling languages to determine if they are semantically similar? What method should be employed? One suggestion is to lay down two models, describing the same thing, next to one another and try to perform a mapping of the different symbols. The first symbol (if it is possible to establish what symbol is the first) in language 1 is mapped to the first symbol in language 2, and so on.

This method resembles the method of placing two books next to each other and start mapping the symbols in the books. If a symbol in the first book is always mapped to the same symbol in the other book, and vice versa, and the order in which the symbols are presented in both books are the same, then the books are deemed to be similar.

To be able to do this one have to start by establishing what the basic symbols are, the alphabet, and how they can be legally combined. Is it the letters that are the basic symbols? Or perhaps the words? Sentences? Paragraphs? Let's assume that the alphabet consists of words, i.e. strings of letters. For instance, these words could be: "send" and "receive". Then the first word in book 1 is mapped onto the first word in book 2, the second word in book 1 is mapped to the second word in book 2, and so on. One immediately realises that this approach is not feasible. To draw the conclusion that the two books are similar in content is almost impossible. The method might work on the odd occasion where the book is very short (perhaps only one word) or just by sheer luck, but it will generally fail.

A better approach is to agree on or determine what exist in reality and consider the aspects that are of interest. What is determined are the things that exists, how they change and behave, if they are complex or simple, and their relations to other things. The model of common knowledge about what exists in reality, is called an ontology model or (for short), an ontology. The constructs in the ontology are the concepts of reality and an ontology

is sometimes also called a conceptual model. When doing comparisons of models expressed in different languages, the concepts of the ontology are used as a reference model.

A comparison of two models made in two different process modeling languages could be done like this: for each concept in the ontology the corresponding construct in each of the models are looked for and identified. If we find that all constructs in the first model has corresponding constructs in the second model and that all constructs in the second model has corresponding constructs in the first model, then we can conclude that the model made in the first language and the model made in the second language describe the same thing.

For example, if we are able to determine that book 1 start by in some way expressing the concept 'send' and this is followed by ('follow' is also a concept, as are 'start' and 'end') the expression of the concept 'receive' and that book 2 expresses the same concepts, and vice versa, then we can be quite confident when we draw the conclusion that the two books are semantically similar. They may look very different but that is no more strange than that the same book look different when printed in English or Chinese.

Using this method, where we focus on the meanings of the models to be compared, gives us a far better opportunity to do good comparisons of models constructed in different languages regarding their similarity, rather than using the method of direct mapping of symbols. The idea, that we should focus on the meanings when comparing, not the notational symbols, is the underlying idea in the method presented and investigated in this thesis.

1.5 Method

In this thesis we propose a method for comparing process models that establishes and uses an ontology as a reference model when doing the comparisons. By reference model is meant that the concepts of the ontology is to be used as a norm when settling the meanings of the notational constructs (symbols and symbol combinations) of the models that are to be compared.

A process metamodel is constructed based on the concepts in the ontology. There are two reasons for constructing a process metamodel from the concepts of the ontology: first, if the ontology is very general and all-encompassing focusing on a sub-set of the concepts makes it easier to grasp and manage the concepts. Secondly, a metamodel is a description of a model. The constructs in the modeling languages are described in the respective languages metamodels. By developing a process metamodel based on the ontology and presenting the metamodel in a well known notation, a comfortable basis for comparison with the metamodels of different languages is established. The concepts in the metamodel together with some additional concepts from the ontology make up the reference model.

Next, the process model property to be compared is decided and defined.

It is important that this is done since the models could be compared with respect to different properties or combinations of properties.

Following definition of property to be compared and with the ontology and the metamodel in place, a mapping schema is constructed. A mapping schema is a description that shows what concepts in the reference model are mapped onto what notational constructs in the modeling languages. The mapping should be presented with a textual motivation, and as the process models often uses a graphical notation, some means for describing what concept maps onto what graphical construct should be provided. The reason for this is that constructing a mapping schema is very difficult and the work done should be recorded as clearly as possible to make it easy to follow the line of thought, and thereby minimise misunderstandings and errors.

The final step of the method is to validate and justify the mapping schema. This is done by using it when doing real comparison. The knowledge gained, when errors are discovered in the comparisons, are used to refine and further justify the schema. This imply that a highly iterative way of working when constructing and using the mapping schema is necessary. A typical scenario is that a mapping schema is constructed, it is validated, some weaknesses are found when validating it, this is recorded textually and the schema is justified. A new validation takes place, new errors surface, they are recorded, the schema is further justified, and so on.

In short, the method for doing comparisons to be investigated is this:

- A domain ontology is established
- A process metamodel is constructed based on a the concepts in the ontology
- A definition of the property to be compared is made
- A mapping schema is constructed where concepts in the ontology and metamodel is mapped onto constructs in different process models
- Validation and justification of the mapping schema is done by performing real comparisons.

1.6 Investigation approach

1. An ontology was chosen, studied and described. The ontology chosen was the ontology originally suggested by Bunge[3] and later on was extended by Wand and Weber[4][5]. The reason for choosing this particular ontology is that it was well defined and was at the time of choice, readily available. Frisco[6] was one other candidate ontology not chosen for the reason that it, at the time, appeared to be quite similar to the chosen one.

2. A metamodel was constructed. The metamodel was constructed from concepts of the ontology and presented using the UML Class diagram notation. The reason for choosing the UML notation was that it is widely used and understood.
3. The property that was to be compared was defined. In this case the property defined was semantic similarity.
4. Two process modeling languages was chosen. The languages chosen, SDL and BPM, were of different kinds to make the comparisons more interesting. SDL was chosen because it has some properties that were of interest in an ongoing research project and BPM was chosen because it was readily available.
5. A modeling task was defined. The task was not too difficult, which would have resulted in spending too much time on preparation and analysis of the models. But it was not too easy which would have made it useless for drawing any conclusions.
6. The task was modeled. The task was first modeled in SDL and subsequently in BPM.
7. A mapping schema was constructed. The schema was created where the concepts of the ontology and the metamodel were mapped onto notational constructs of SDL and BPM. It was not possible to use only the available definitions and descriptions of the languages to make the mappings. To be able to do the mappings, the task models were used to determine the semantics of the language constructs.
8. One comparison and one translation was made. The comparison was made in the following manner: one process from the SDL task model was chosen to be compared to the same process in the BPM model. The SDL process was mapped onto the concepts provided in the mapping schema. After this, a BPM model was constructed based on mapping result. This newly constructed BPM model was subsequently compared to an already existing BPM model of the same process. A conclusion about their similarity was drawn according to the definition of similarity. A translation was carried out in the following way: one process from the BPM task model was chosen to be translated. The BPM process was mapped onto the concepts provided in the mapping schema. The concepts describing the BPM process were then used when constructing the SDL process model. Using the definition of process similarity, a conclusion could be drawn about the similarity of the BPM and SDL processes.

1.7 Intended audience

The intended audience for this thesis is anybody who is interested in a brief introduction to process models and a method for comparing them. Some previous knowledge on the topic of modeling is assumed on behalf of the reader.

1.8 Thesis scope

In this thesis we argue that making comparisons and translations are to be considered as being tasks of the same kind. Both comparisons and translations involves capturing the syntax and the semantics of the modeling language.

In its original form, the ontology introduced it is given in a highly formalised set-theoretic language which is hard to read and understand for anyone who is not trained in reading such formalisms. The ontology introduced is presented in a way that is readable and understandable for the intended audience. This means that the original high level of definitional precision is decreased.

2 Theoretical Framework

2.1 Ontology

2.1.1 Introduction

Ontology is a well-established theoretical domain within philosophy dealing with models of reality.

Wand and Weber have taken, and extended, an ontology presented by Bunge and applied it to the modeling of information systems. Their fundamental premise is that any information systems modeling language must be able to represent all things in the real world that might be of interest to users of information systems; otherwise, the resulting model is incomplete. If the model is incomplete, the analyst/designer will somehow have to augment the model(s) to ensure that the final computerised information system adequately reflects that portion of the real world it is intended to simulate.

The Bunge-Wand-Weber (BWW) models consist of the representation model, the state-tracking model, and the good decomposition model. This thesis focuses on the representation model. The representation model defines a set of constructs that are thought by Wand and Weber to be necessary and sufficient to describe the structure and behaviour of the real world. The following presentation is intended to be an intuitive introduction to the concepts.

2.1.2 Thing, Property, Attribute, State, Law

The world is made out of *things* that possess properties. There exist simple things and simple things can associate to form composite things. If things X, Y, \dots , associate to form a thing T , then X, Y, \dots , are each a component of T .

Properties can be intrinsic or mutual to several things. For example, having weight is an intrinsic property of a person. Being employed is a mutual property of a person and a company. A property of a composite thing can be either inherited—be a property of a component, or emergent—be a property of none of the components. A composite thing must possess emergent properties. Emergent properties depend on the properties of the components, but are not necessarily reducible to them. In other words, an emergent property may not be simply replaced by component properties, although it might be explained by them.

A thing possesses properties whether humans are aware of them or not. In contrast, *attributes* are characteristics assigned to things by humans. For example, people will attribute a colour to an object while the real property is the reflection of some wavelength. An attribute can be represented as a function from a set of things and a set of “observation points” (in particular, time points) into a set of values. This is the basis for defining a model of a thing as a functional schema which is a set of attribute functions defined over a certain domain, M (usually time). The *state* of a thing comprises the values of the functions in the functional schema at a given point in M . Similar things can be modeled using the same functional schema.

Every property can be modeled as an attribute but not every property will be described in terms of attribute(s). The same property might be represented by more than one attribute. For example, the power of a motor can be described in more than one way (horsepower, torque at certain RPM's, etc.). Several properties might map into one attribute. For example, the notion of IQ reflects many properties (not necessarily known). Moreover, not every attribute has to represent a property. An example of this is the name of a person.

The properties of a thing include *laws*, which are constraints on other properties (and can specify relationships among properties). For example, in the employee example mentioned above, there could be a law connecting rank and salary. Because of laws, not all possible combinations of state function values represent valid states of a thing. Laws are modeled as restrictions on the possible states.

2.1.3 Event, Transformation, Behaviour, History, Acts on

It is an ontological principle that every change is tied to things and every thing changes. When a thing changes, some of its property values must

change. Change can be modeled as a state transition—that is, as an event. An *event* is a triplet $\langle s1, s2, g \rangle$ where g is one or more *transformations* that changes state $s1$ to state $s2$ and conforms to the laws of the thing. The transformations that can occur inside a thing can be described in terms of a transition law which specifies the possible state transitions

For the purpose of modeling dynamics of systems, several concepts have been defined in addition to those of Bunge, but based on his concepts. Specifically, there is a distinction between two types of state changes. Change of state due to the actions of other things, i.e. external events; change of the state due to transformations inside the thing, i.e. internal events.

One distinguish between two types of states, stable and unstable. A thing can change state from a stable state only due to an external event—some external thing cause the change from the stable state. A change from an unstable state do not need interaction with an external thing to happen. A thing can change state from a stable state only due to an external event and so, in order to analyse the behaviour of a thing, the external events that can happen to it should be known.

The *behaviour* of a thing comprises its change of state due to external and internal events. Assume a thing is in a stable state. It will stay in this state until an external event happens. The external event can result in a change to a stable state or an unstable state. In the latter case there will follow an internal event and the thing will change to yet another state, which, in itself, could be stable or unstable.

Changes of state manifest a *history* of a thing. The history is a recording of property value changes of a thing at time instants. The notion of history allow us to determine when two things are coupled to each other. Intuitively, if two things are independent of each other, they will have independent histories. If they are coupled in some way, however, at least one of the things history will depend on the other things history. A thing *acts on* another thing if it affect the other things history. Two things are coupled if the first one acts on the second one, or if the second one acts on the first one, or both.

2.1.4 System, Subsystem, Composition, Structure

Two things interact if at least one of them can affect the states the other traverses in time. The ability of two things to interact is a mutual property of the two things. In other words, there is no “acting at a distance”. Interaction can be modeled via joint state variables (representing the mutual properties).

A *system* is defined as a composite thing made of interacting things. The *composition* of a system is the set of components in the system. The environment of the system comprises of the things not in system that interact with components of the system. The *structure* of the system is the set of interactions that exist among components in the system and between

components in the system and things in the environment. Identifying the composition of a system requires that a level of detail be specified. Since a system is a composite thing, it has emergent properties. In particular, the dynamic properties (i.e. the behaviour) of the system emerge from the behaviour of its components, and their interactions.

The definition of a system does not specify the notions of a goal and a boundary that are commonly mentioned in the context of systems. The concept of goal is subsumed into the transition law of the system as “preferred” states to which the transition leads. Similarly, there is no explicit notion of a boundary. Instead, there are the composition and the set of relevant external events. The composition defines what is inside the system and the relevant external events define what happens outside the system that affect the system.

2.1.5 Comments

Some concerns have consistently arisen[6] within the research community over the time that the BWW ontological models have been developed and used.

The understandability of the ontology has been criticized. Wand and Weber originally defined the concepts using a rigorous set-theoretic language. Even though they have attempted to simplify and clarify the explanation of the concepts by defining them using plain English, the criticism of lack of understandability has remained.

There is a difficulty in applying the ontology, e.g. when comparing models or translating between languages. This difficulty stems from the fact that, although the BWW concepts have clearly defined definitions, the models are constructed in languages that sometimes, at best, have loose definitions. Consequently, the analyses performed using the BWW models rely to a large extent on the knowledge and experience of the researcher(s) performing the analyses.

2.2 A process metamodel

To address the problems of understandability and applicability, and also focus on the process parts of the BWW ontology, an attempt is made at providing a semi-formal description of some of the core BWW concepts by designing a process metamodel. What is required for the concepts is a definitional approach that is familiar to users, more specific than plain English, but easier to understand than set-theoretic language. By using a relatively well-known language (UML class diagrams, in this case), the current understanding of the concepts and how they relate to each other can be explained quite clearly.

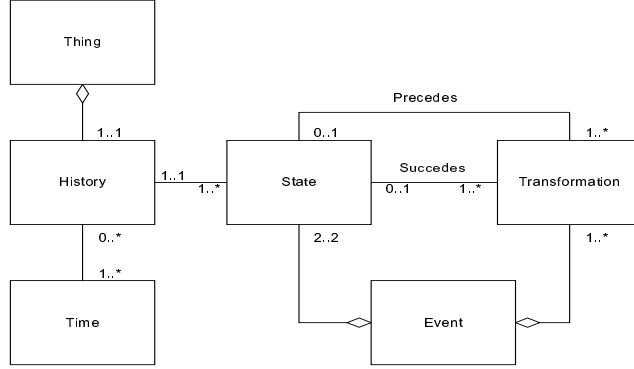


Figure 1: Process metamodel

The approach of constructing a metamodel has some additional advantages: first, the developed metamodel that describes and clarifies the current understanding of the ontological constructs facilitates the use of the ontology in other related research areas. In other words, the assumption is that the availability of an easy-to-understand description of the ontology simplifies the communication about this approach. Second, this approach will also help to identify and eliminate inconsistencies and anomalies within the ontology. Finally, the meta model contribute to the research on the usefulness of metamodeling.

Figure 1 shows a process metamodel which is grounded in the BWW ontology. The metamodel was designed by Rosemann and Green[7]. The notation is the UML Class diagram notation. It shows the concepts that constitute a process. A metamodel like the this one can be regarded as a reference metamodel and the process pattern in this model is a structure that might be expected to exist in every process modeling language.

2.3 Definition of process similarity

The behaviour of a thing can be modeled as a process and this model is described by use of symbols and rules of a process modeling language. When comparing processes, it is important that we understand clearly what we mean when we say that two processes are similar. We start this section that eventually will end up in a definition of process similarity, with a discussion on the definition of process. A quick and dirty definition of a process is this:

A process is a partially ordered set of processes

This needs to be clarified.

To define the behaviour of a thing from a basic level amounts to this: Assume that a thing has some outer, functional characteristics. That is, let us think of it at a “black box”. The thing is to be thought of as an automate whose inner structure it is not necessary to be known, but can be assumed to give well-defined responses to well-defined stimuli.

Let us call one such “black box” an activity. A process is then defined to be an ordered set of one or more activities. The reason for making this reformulation is to point out that a process can be modeled at different levels of abstraction. Our intention, when calling a process an activity, is that it is not possible or necessary to analyse it further. So, we reformulate the definition of process given above to the equivalent, but clearer:

A process is a partially ordered set of one or more activities

Some activity in a process is the first and some is the last. If the process only consists of one activity, the first and the last are the same. Certain conditions must be true to activate or start a process. The conditions for starting a process are identical to the conditions for starting the first activity in the process. The conditions to start a process is made true by some activity outside the process. When the conditions for ending the process are made true by the last activity in the process, the process stops. Some parallel or concurrent activities can take place within a process as long as they do not interfere with each others performance. This restriction is for simplifying the analyses. All activities, concurrent or not, must be finished before the last activity of the process can start.

We define activity similarity which we also may call “outer similarity”:

Two activities is considered to be similar when they always produce the same response to the same stimuli.

This means that we do not concern ourselves with how the results are produced. The production work can use up different amounts of resources, such as space, time, or whatever. As long as the end results are the same, the activities are similar.

From this we move on to define processes similarity which we also may call “inner similarity”:

Two processes are similar if:
they have the same number of activities,
the activities are performed in the same order,
the corresponding activities in the processes are similar

This means that for two processes to be similar they must consist of the same number of activities, performed in the same order, and the result of each activity as well as the results of the whole process must be similar. For example, if two processes show the same process results but the results are produced by a different number of activities, then the processes are not similar by definition.

3 Process Modeling Languages

3.1 Introduction

It is possible to consider a process as consisting of two logically different types of activities: First, there are the activities that does the “real” work in a process, such as “heating the water” in a coffee making process, or “creating an account” in some banking process. Secondly, we have the governing activities that control and coordinate the “work performing” activities according to the rules and laws to which the process must comply. These are the activities that make sure that the water, in a coffee making process, is heated before it is filtered through the coffee.

We can, when we are about to model a process, start by concentrating on one type of activity first and later add activities of the other type.

If we want to model a process by first and foremost show the “work performing” type of activities that is part of the process and how these activities are related, we describe a sequence of such activities where the finishing conditions for one activity is identical to the starting conditions for the next activity. The activities that are of the controlling and coordinating type are only shown implicitly through the construction of the “work performing”-activity sequence where one activity is to be finished before another. However, controlling type of activities like selection of process paths or iterations are usually shown explicitly. The main idea in this approach is that we concentrate on modeling the process by describing it as a sequences of activities, its activity-flow.

If we, on the other hand, want to model a process by showing how it is controlled and coordinated, some other modeling strategy should be employed. We can think like this: a controlling activity receives information

stating that the finishing conditions of some “work performing”-activity, either in the process or in the process environment, has become true. Then, according to the rules that govern the process and of which the controlling activity are aware, the controlling activity starts the next “work performing”-activity by sending information to it stating that the conditions for it to start are true. When this next activity is finished (the conditions for its finishing are true), a controlling activity is notified and the cycle can start all over again. The essence of this approach is that we model the process by describing the communications that takes place between its parts, its control-flow.

The approaches are equivalent in the sense that both describe what happens in a process and when. However, the second (communications) approach tends to introduce slightly more details into the models.

The two approaches provides an opportunity to classify some process modeling languages according to their underlying model. The first approach, concentrating on describing the flow of activities, has as underlying model an event-process chain (EPC). The second approach, concentrating on describing the control-flow, has as underlying model a finite state machine (FSM).

In the following, two process modeling languages representing each of the two underlying models will be presented. BPM (Business Process Modeling)[8] represent an EPC, SDL (Specification and Description Language)[9] represent a FSM. The descriptions of the languages presented below are by no means complete but sufficient to get an understanding on how they are used.

3.2 BPM

BPM (Business Process Modeling) is mainly targeted at business managers wanting to describe and improve business processes. BPM as presented here is a subset of the core concepts, adapted by Allen and Frost[8]. The BPM notation is shown in figure 2 (lower).

3.2.1 Concepts

The three main main concepts are elementary business process, business event, and process thread.

An elementary business process (EBP) is an activity performed by one person in one place at one time, in response to a business event.

A business event is a stimulus that trigger a EBP. Business events may be of two different kinds: input or output driven. Input-driven business events are signaled by the arrival of incoming information flow; they can be external (produced outside the system), or internal (produced inside the system). Output-driven business events are may be temporal or conditional. Temporal events are are signaled by the arrival of a predefined point in time. Conditional events reports the sensing of a particular circumstance, which triggers an EBP.

A process thread is a chain of EBP's. It is used to model the flow of EBP's, initiated by a business event in terms of sequence dependency, iteration, conditionality and process breaks (waitings for something to happen). A process thread normally produces a result. A result from one process thread is often a business event relative to some other process thread. Process threads can be diagrammed at a higher level of granularity by clustering EBP's into process groups.

The Process Hierarchy Diagram is used mainly as a graphical index, which shows the relationship between the levels of process granularity. At the very top sits the enterprise (or whatever under consideration). This is divided into key business processes which in turn may be divided into process groups, and so on. Eventually one reaches the level of single EBP's although stopping at a group level is quite normal. The Process Thread Diagram is the major modeling tool. It is used to model dependencies between EBP's.

3.3 SDL

SDL (Specification and Description Language) was developed by telecommunications companies to cope with the increasing complexity of their systems. The purpose of SDL is to provide an unambiguous specification and description of a systems behaviour. The intended users are mainly technicians and engineers. Presented here is a subset of concepts and symbols in the language.









The relevant SDL notation is shown in figure 2 (upper). Note, in order to minimise the number of symbols and keep the models compact we shall use the following diagrammatical convention: a process in a SDL model will be considered to have finished when it returns to its first state (which is designated S1 in all models).

3.3.1 Concepts

A system is described as a number of extended finite state machines communicating by exchanging messages with each other and with the environment of the system. All communications between the system and its environment and within the system is achieved by sending messages. The state machines may work in parallel, and several instances of the same state machine may exist concurrently. The extended finite state machine is modeled by the process concept.

The behaviour of a process is described by a start node, which is the state the process is in when it is started, and a set of transitions that takes place when the process receives a certain stimulus in a certain state. Part of a transition may be the output of one or several messages and after finishing a transition the process enters a new state. A message is modeled by a signal that may carry a list of values. The values are of some type. When sending a

SDL Notation

Symbol				
Name	Start	State	Nextstate	Input
Symbol				
Name	Decision	Procedure	Process	Output

BPM Notation





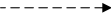

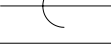
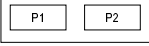
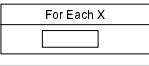

Name	Symbol	Description
Internal Business Event		Internal stimulus which triggers a process thread or activity
External Business Event		External stimulus which triggers a process thread or activity
Result		process
Mandatory Dependency		Leads to a mandatory process
Optional Dependency		Leads to a process to indicate its optionality
Activity		Generic Process
Exclusivity		To indicate a condition
Parallel Processes		containing box to be complete
Iteration		To indicate a repeating process or processes
Process Break		A delay prior to another process starting(ends with an event)

Figure 2: SDL and BPM notation

signal one may directly specify the process instance for which the message is intended or leave it to the system to identify the receiving process instance.

Communication in SDL is asynchronous. No synchronisation is required between sender and receiver. The rationale behind the asynchronous communication is to model a distributed and loosely coupled system. To each

process instance there is an unlimited buffer to hold all received signals not yet consumed by the process. The buffer is called the input port of the process instance.

A Block Diagram describes a system as a set of subsystems and the communications channels between the subsystems and the environment and between subsystems. The subsystems can be decomposed further to reach an appropriate level of detail. A Process Diagram shows the behaviour of processes in a block.

4 Modeling exercise

4.1 The coffee machine

At work, there is a coffee machine. I know that it is a coffee machine because someone in past times, while nodding at it, told me so. It is called “the coffee machine” although it produces other beverages as well. A description of a very simplified version of the machine is as follows:

The coffee machine is a brown-colored box with three buttons and a place to put a cup at the front. The first button is labeled “Normal” , the second button is labeled “Strong”, and the third button is labeled “Chocolate”. By placing a cup at the right place and pressing a button it will fill the empty cup with the beverage of choice. It is not necessary that there is a cup present. It will produce the beverage regardless of the cup being present or not. If anything extra is needed, like sugar, then the machine cannot help. Boxes with sugar and other things are kept beside it.

I do not know how the machine produce these beverages, but I define it to be organised and to operate as follows: The machine consists of five general parts. First, it has a casing, holding the other four units. Secondly, it has a unit responsible for controlling and coordinating the activities. Thirdly, it has a unit responsible for portioning variable amounts of beverage powder. The fourth unit is a unit responsible for applying filters. And finally, it has a unit for handling the water. It is responsible for heating and pouring a volume of water.

Here’s how the machine brew a cup of coffee: The controlling unit receives an activation signal from one of the coffee buttons. It makes the filter unit apply a filter and makes the water unit heat a volume of water. When the filter is applied, coffee is put into the filter. After this, water is poured through the filter thereby producing a cup of coffee.

Making a cup of chocolate is done almost in the same fashion. The control unit receives an activation signal from the chocolate button. It makes the water unit heat a volume of water and makes the beverage powder unit portion chocolate and milk powder straight into the cup. After this, water is poured into the cup and the chocolate is ready.

When producing a normal cup of coffee, the machine uses one standard amount of powder. When producing a strong cup, the machine uses two amounts. When producing chocolate, the machine uses one standard amount of chocolate powder and one standard amount of milk powder. The amount of water is the same for all three beverages.

4.2 Models

Models of the coffee machine processes are found in the appendices. The BPM models are found in appendix A. The SDL models are found in appendix B. The models are placed in the appendices because there are so many of them.

4.3 A mapping schema

This is the first attempt at constructing a mapping schema that can be used for doing comparisons and translations. Making a correct schema is quite difficult so this work should be carried out in a highly iterative manner. That is, a schema must be tried and tested in a real modeling situation and when it fails, the failures must be amended. The mappings motivated in this text are presented in a graphical table found in figures 3 and 4.

Thing: A thing is some distinct logical and/or physical unit that exhibits some behaviour. Thing is modeled as a named rectangle in both SDL and BPM and are shown in the static diagrams of SDL and BPM. The other constructs in the static diagrams, System, Subsystem are interpreted to have direct correspondences. System structure in SDL has no corresponding construct in BPM.

Property -mutual: Communication is modeled through change of mutual properties. In SDL, communication is modeled by signal symbols. The name inside such a signal symbol is the name of the mutual property. It can be thought of as a variable in the receiver being changed by the sender. In BPM, this is shown as the name within an event arrow with a thin arrow emanating from it. This is a name of a property that has been changed by an activity and this change imply the start of a new activity.

Property -intrinsic: In SDL, this is shown as a name within a rectangle. The rectangle has also the keyword DCL written inside it. Changing an intrinsic property does not imply change in another thing. The intrinsic property can be thought of as a local variable. In BPM this is shown as the name within an event arrow with no thin arrow emanating from it. This is a name of a property that has been changed by an activity and this change does not cause the start of a new activity.

Acts on: Indicates that a thing will cause an reaction in another thing, a kind of dependency. In SDL this is shown by the sending of a message to a receiver. In BPM this is shown by a thin arrow from an event-arrow

Mapping Schema














Concept	SDL (Process Diagram)	BPM (Process Thread Diagram)
<i>Property-mutual</i>	 <i>Name within output/input-signal symbol</i> 	 <i>Name within big arrow with thin arrow or half-ellips with thin arrow</i> 
<i>Property-intrinsic</i>	 <i>Rectangle with reserved word 'DCL' and name of property</i>	 <i>Name within big arrow or half-ellips</i> 
<i>State</i>	 <i>Ellips with name</i>	
<i>Transformation</i>	 <i>Rectangle with two extra lines and name</i>	 <i>Rectangle or half-ellips with name</i> 
<i>Acts on</i>	 <i>Output signal symbol</i>	 <i>Big arrow with thin arrow</i>

Figure 3: Mapping schema 1 of 2

to an activity. This thin arrow indicates that the finishing event of the first activity is identical to the starting event of the second activity. Thus, the first activity has to be finished before the second can start.

Transformation: A transformation is the work that is done to transform a thing. To transform a thing is to change some property of that thing.

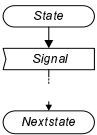
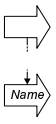
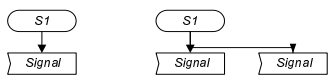




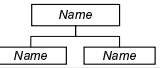
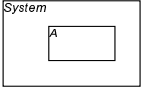
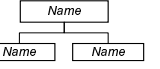
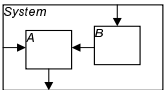
Concept	SDL (Process Diagram)	BPM (Process Thread Diagram)
Event	 <p>Two consecutive ellipses with name and at least one input symbol with name</p>	 <p>Two consecutive big arrows, the second with name</p>
Law	 <p>Single thin arrow emanating from state symbol indicating sequence</p> <p>Multiple thin arrows emanating from a state symbol indicating choice</p>	 <p>Single thin arrow emanating from state symbol indicating sequence</p> <p>Multiple thin arrows emanating from a transformation symbol indicating choice</p>
Concept	SDL (Block Diagram)	BPM (Process Hierarchy Diagram)
Thing	 <p>Rectangle with name</p>	 <p>Rectangle with name</p>
System	 <p>Rectangle with reserved word 'system'</p>	 <p>Rectangle connected to and above other rectangles</p>
Subsystem	 <p>Rectangle with name inside system or other rectangle</p>	 <p>Rectangle connected to and below other rectangle</p>
System structure	 <p>Rectangle with name inside system or other rectangle connected with thin arrows</p>	

Figure 4: Mapping schema 2 of 2

In SDL this is shown by a procedure-symbol, in BPM this is shown by an activity symbol. Procedure and activity maps directly onto each other.

State: The state of a thing comprises the property values of the thing at a point in time. In SDL this is shown by an ellipse with a name. An SDL state is not mapped onto anything in BPM. However, concerning the

mapping of states in SDL onto BPM it is possible to argue like this: In SDL, a transformation of a thing means that a thing has gone from a state before the transformation to a state after the transformation. A procedure transforms a thing between two states. In BPM, a transformation of a thing means that it has passed from the starting event of an activity to the finishing event of the activity. An activity transforms a thing between two events. So, the places where the event symbols in BPM are drawn corresponds to where the states are in SDL.

Event: An event is a change of states together with the transformation(s) that are necessary in order to change the state. In SDL, an event is shown by two named ellipses connected by arrows and other symbols. Most notably among those other symbols are the in-signal symbol indicating that the conditions for the start of the event are true, and any number of procedure symbols according to the definition of the event. In BPM, two event arrows connected by thin arrows and activity symbols comprise an event.

Laws: Laws show in what order the activities in a process should be performed and under what conditions they should start and finish. Illegal sequences of activities are not modeled. Laws are modeled in the same way in both BPM and SDL.

4.4 Comparison and translation

In this section a comparison the process of making a normal strength cup of coffee is made. This process is the only one compared. The other processes can be analysed in an analogous way.

4.4.1 Comparison

The comparison of two process models made in different modeling languages is made in the following way: a process in one model is described in terms of the concepts in the mapping schema. The description made is used to make a model of the same process in the other language. The resulting model are used when comparing it to another model of the same process made in the same language. When this is done a conclusion about their similarity is drawn.

In this example, we start by describing the SDL process in terms of the the reference model concepts in the mapping schema. The description is used to construct the process in the BPM notation. The simplified version of the original SDL model is shown in figure 5 (left) and the constructed BPM model is shown in figure 5 (right). The BPM model is used later in the comparison.

1. Due to the occurrence of an event in the environment of the process which result in change of mutual property 'Make_bev(Choice)', an event in process thing 'controlunit', that require the change of mutual

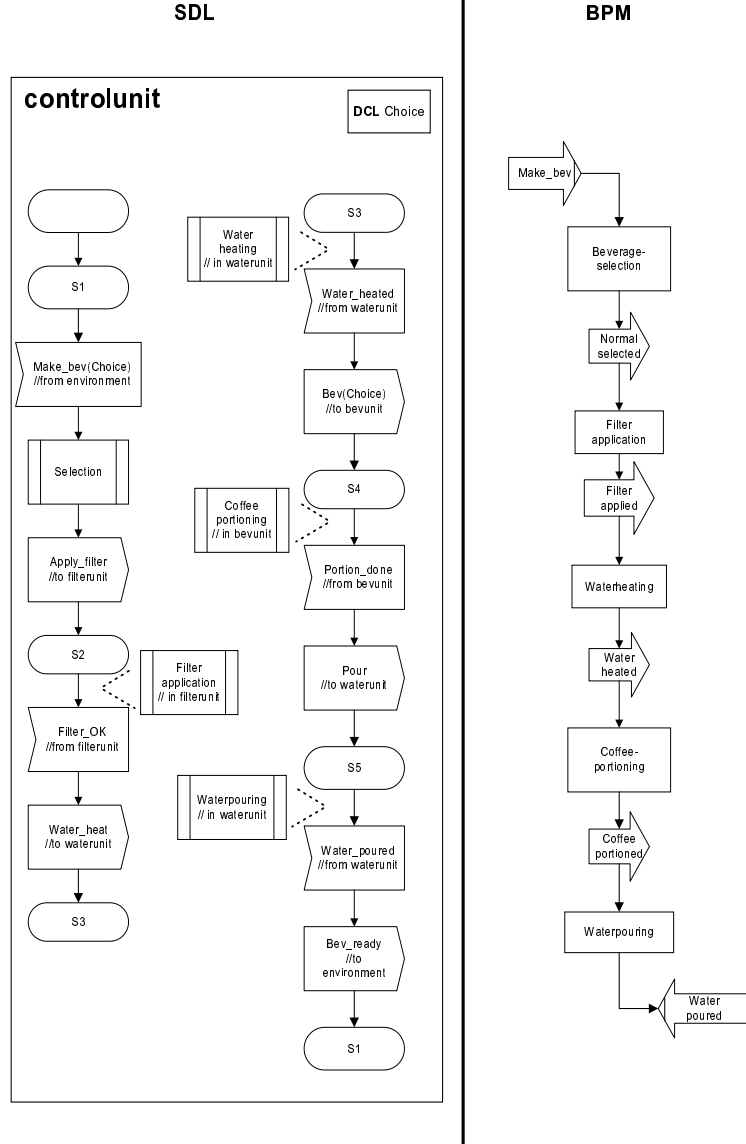


Figure 5: Comparison of SDL and BPM

property 'Make_bev(Choice)' occurs. The occurrence of the event in 'controlunit' result in change of mutual property 'Apply_filter' and intrinsic property 'Choice'. By changing mutual property 'Apply_filter', 'controlunit' acts on process thing 'filterunit'.

2. Due to the occurrence of an event in process thing 'filterunit' which

result in change of mutual property 'Filter_OK', an event in process thing 'controlunit', that require the change of mutual property Filter_OK occurs. The occurrence of the event in 'controlunit' result in change of mutual property 'Water_heat'. By changing mutual property 'Water_heat', 'controlunit' acts on process thing 'waterunit'.

3. Due to the occurrence of an event in process thing 'waterunit' which result in change of mutual property 'Water heated', an event in 'controlunit', that require the change of mutual property 'Water heated' occurs. The occurrence of the event in 'controlunit' result in change of mutual property 'Bev(Choice)'. By changing mutual property 'Bev(Choice)', 'controlunit' acts on process thing 'bevunit'.
4. Due to the occurrence of an event in process thing 'bevunit' which result in change of mutual property 'Portion_done', an event in 'controlunit', that require the change of mutual property 'Portion_done' occurs. The occurrence of the event in 'controlunit' result in change of mutual property 'Pour'. By changing mutual property 'Pour', 'controlunit' acts on process thing 'waterunit'.
5. Due to the occurrence of an event in 'waterunit' which result in change of mutual property 'Water_poured', an event in 'controlunit', that require the change of mutual property 'Water_poured' occurs. The occurrence of the event in 'controlunit' result in change of mutual property 'Bev_ready'. By changing mutual property 'Bev_ready', 'controlunit' acts on the environment of the process.

4.4.2 Result of comparison

The BPM model created is in figure 5 (right). Let us compare the BPM model of the process of making a normal strength cup of coffee from figure 5 to the corresponding process thread in the BPM model from appendix A, figure 7 , by placing them next to each other. The thread "Normal" is the middle of the three threads. As we can see, they model different processes. The activities are clearly not performed in the same order. Specifically, the SDL process is a strict sequence of activities but the BPM process contains concurrent activities. So, by the definition of process similarity the processes are not similar. However, they are similar in the sense that both processes produce a normal strength cup of coffee when the "Normal" button is pressed. The processes do not show process similarity, but shows activity similarity as it is defined in section 2.3.

4.4.3 Translation

This is a comparison of two models, where only one exists from the start. First, we need to construct the non-existing one. In doing this construction

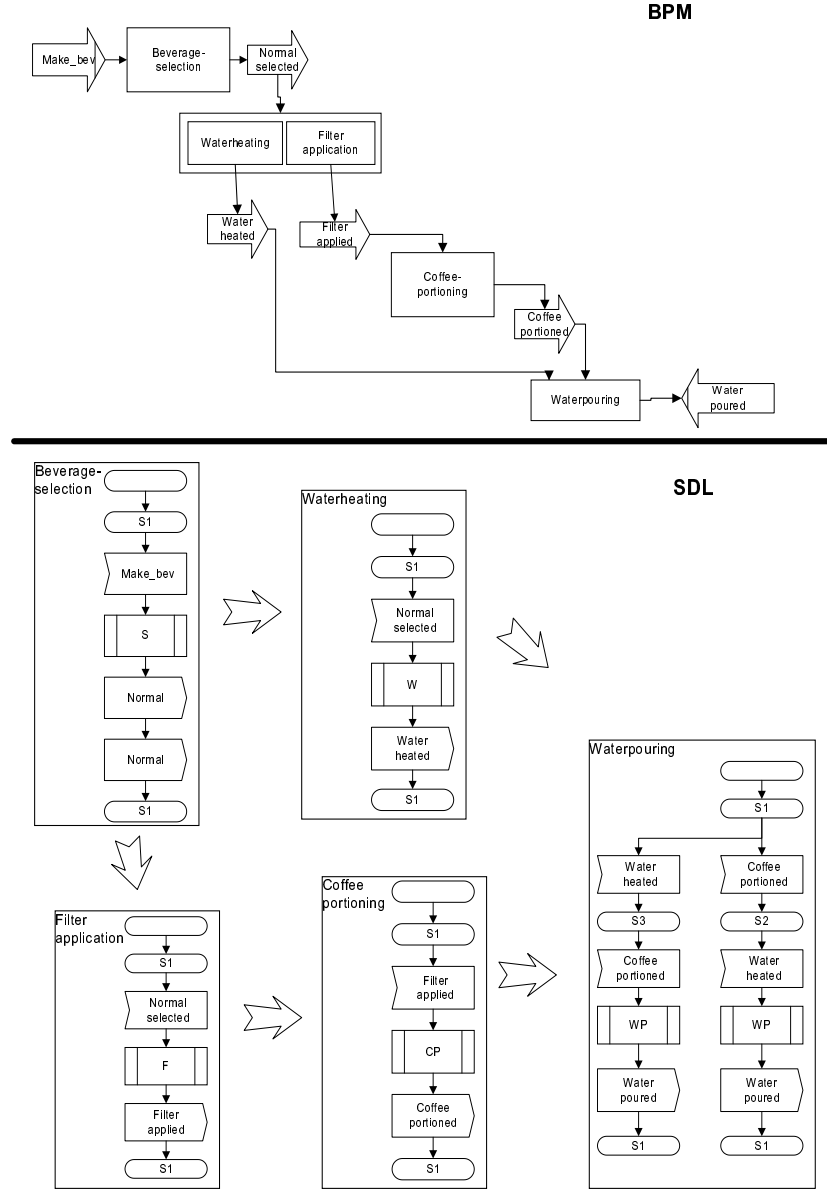


Figure 6: Translation from BPM to SDL

we can say that we translate the first model into the other. What we can expect from a translation is that the resulting models are similar. In this example we translate a model from BPM to SDL. The translation is made in the following way: The process thread “Normal” from the BPM diagram is described in terms of the reference model concepts in the mapping schema.

Subsequently, this description is used when constructing a SDL model of the same process thread. The SDL model constructed is presented in figure 6 (lower), where the original BPM process thread is presented in figure 6 (upper).

1. Due to the occurrence of an event in the environment of the process, which results in a change of mutual property 'Make_bev', an event in process thing 'Beverage selection' that requires the change of mutual property 'Make_bev' occurs. The occurrence of the event in 'Beverage selection' results in a change of the mutual property 'Normal selected'. By changing mutual property 'Normal selected', 'Beverage selection' acts on both process things 'Waterheating' and 'Filter application'.
2. Due to the occurrence of an event in the process thing 'Beverage selection', which results in a change of mutual property 'Normal selected', an event in process thing 'Waterheating' that requires the change of mutual property 'Normal selected' occurs. The occurrence of the event in 'Beverage selection' results in a change of the mutual property 'Water heated'. By changing the mutual property 'Water heated', 'Waterheating' acts on process thing 'Waterpouring'.
3. Due to the occurrence of an event in the process thing 'Beverage selection', which results in a change of mutual property 'Normal selected', an event in process thing 'Filter application' that requires the change of mutual property 'Normal selected' to occur. The occurrence of the event in 'Filter application' results in a change of the mutual property 'Filter applied'. By changing the mutual property 'Filter applied', 'Filter application' acts on process thing 'Coffee portioning'.
4. Due to the occurrence of an event in the process thing 'Filter application', which results in a change of mutual property 'Filter applied', an event in process thing 'Coffee portioning' that requires the change of mutual property 'Filter applied' occurs. The occurrence of the event in 'Filter application' results in a change of the mutual property 'Coffee portioned'. By changing the mutual property 'Coffee portioned', 'Coffee portioning' acts on process thing 'Waterpouring'.
5. Due to the occurrence of events in the process things 'Waterheating' and 'Coffee portioning', which results in a change of mutual properties 'Water heated' and 'Filter applied', an event in process thing 'Waterpouring' that requires the change of mutual properties 'Water heated' and 'Coffee portioned' to occur. The occurrence of the event in 'Waterpouring' results in a change of the mutual property 'Water poured'. By changing the mutual property 'Water poured', 'Waterpouring' acts on the process environment.

4.4.4 Result of translation

The SDL model created in the translation is in figure 6 (lower). Now, let us compare the BPM model and the SDL model by placing them next to each other. We see that the models contains the same number of activities. The activities are performed in the same order. Corresponding activities in the process models are similar. We conclude that, according to the definition of process similarity from section 2.3, the two models describe similar processes.

5 Conclusion

This paper has proposed and investigated a method for comparing process models with respect to their semantic similarity. The method was described and it was validated by trying it in a practical example. The conclusion drawn from the validation is that the method indeed can be used to determine whether two models are semantically similar or not.

Some experiences gathered when using the method in the practical example are; although the SDL documentation was adequate, the BPM documentation was not. This made it difficult to establish the semantics of the BPM notation. The availability of good documentation of the languages used is quite important. The Bunge-Wand-Weber ontology was not easy to use as starting point for modeling and understanding process models. Wand and Weber have concluded from their research that the ontology has deficiencies with respect to dynamics. This makes it less than perfect when used for analysing process modeling.

Future research concerning the method include the following; an alternative ontology, that better handle dynamics should be used. An alternative possible ontology is Frisco[6]. The method should be tried and tested in a more complicated modeling task, perhaps in a business setting. The models to be compared in the should be be made in other languages. Strong candidates when doing this are BML (Business Model Language)[11] and UML Activity diagrams.

References

- [1] Simon, H. A., *The Sciences of the Artificial*, 3rd ed. ISBN 0-262-19374-4, MIT Press, 1996.
- [2] Rumbaugh, J., Jacobson, I., Booch, G. *The unified modeling language reference manual*. ISBN 0-201-30998-X. Addison-Wesley, 1999.
- [3] Bunge, M., *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World*. Reidel, Boston, 1979
- [4] Wand, Y., Weber, R., *An Ontological Model of an Information System*. IEEE Transactions on Software Engineering, Vol. 16, No. 11, November 1990.
- [5] Wand, Y., *Ontology as a foundation for meta-modelling and method engineering*. Information and Software Technology 38 (1996) pp. 281-287. Elsevier.
- [6] Falkenberg, E. et al. *Frisco, a framework of information system concepts*. ISBN 3-901882-01-4, IFIP 1998.
Downloadable from <ftp://ftp.leidenuniv.nl/pub/rul/fri-full.zip>, 2001-06-05.
- [7] Georgakopoulos, D. Hornick, M. Sheth, A. An Overview of Workflow-Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases, 2, (3), pp.119-153, 1995.
- [8] Rosemann, M. Green, P. Enhancing the Process of Ontological Analysis—The “Who cares” Dimension. Paper submitted to the IS Foundations Workshop, Department of Computing, Macquarie University, Australia. 1999.
- [9] Allen, P., Frost, S., *Component-Based Development for Enterprise Systems*. ISBN 0-521-64999-4, Cambridge University Press, 1998.
- [10] *ITU-T. Recommendation Z.100, Specification and Description Language (SDL)*. Published by ITU, 1999.
- [11] Johannesson, P. Perjons, E. *Design Principles for Application Integration*. Downloadable from <http://dsv.su.se/~perjons/caise226.pdf>, 2001-05-07. Also published in 12th Conference for Advanced Information Systems Engineering (CAISE), LNCS, Springer, 2000.

A BPM models

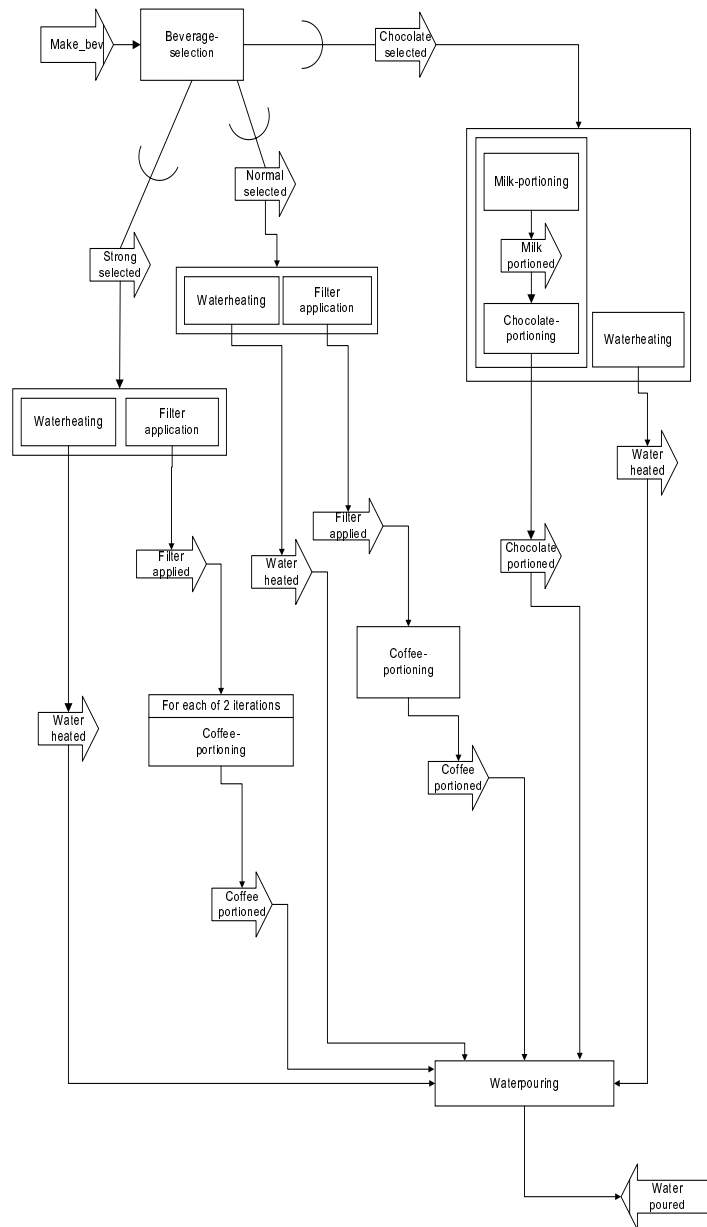


Figure 7: BPM process thread diagram

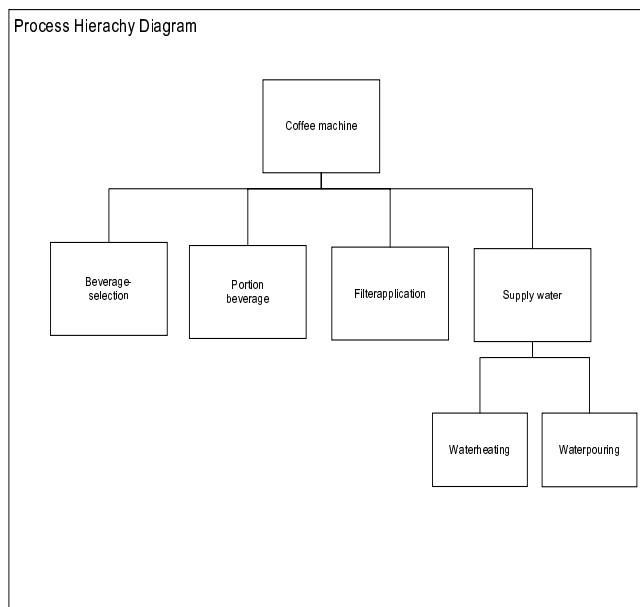


Figure 8: BPM process hierarchy diagram

B SDL models

controlunit

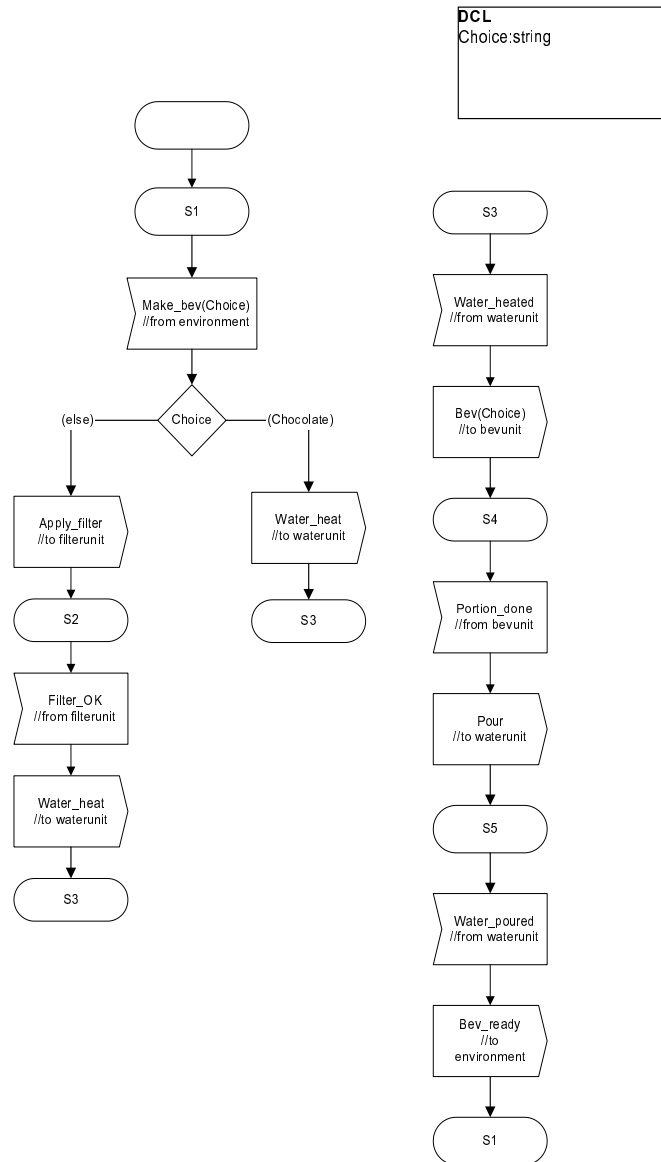


Figure 9: SDL process diagram. controlunit

bevunit

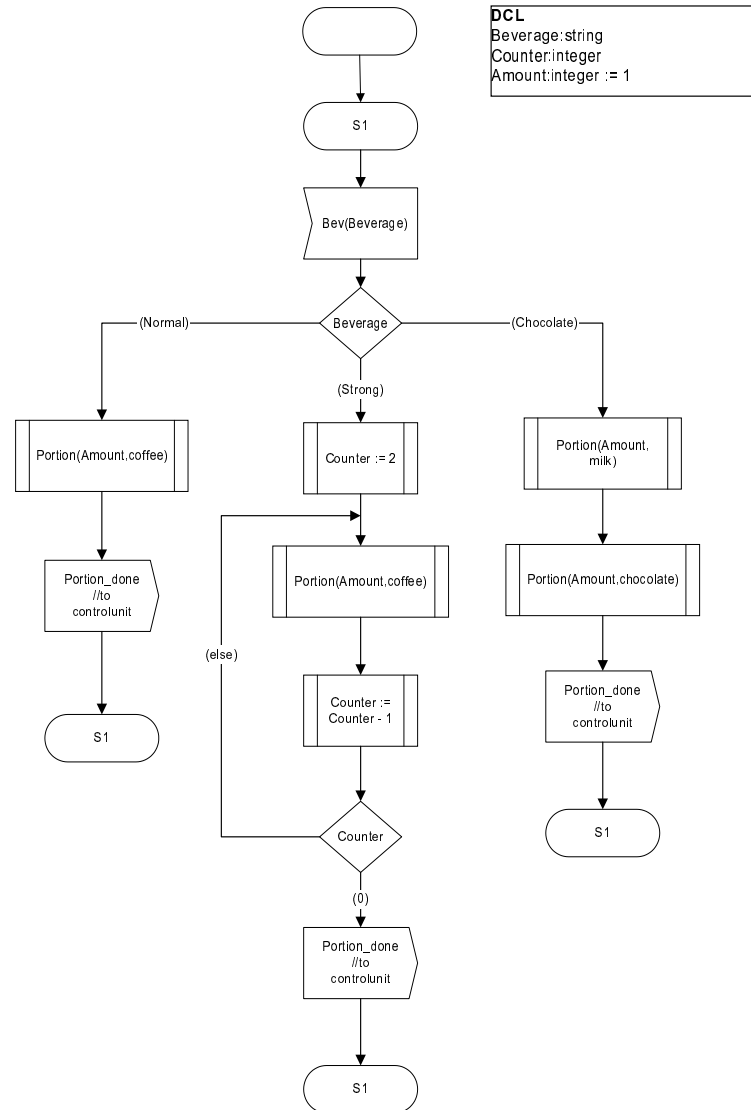


Figure 10: SDL process diagram. bevunit

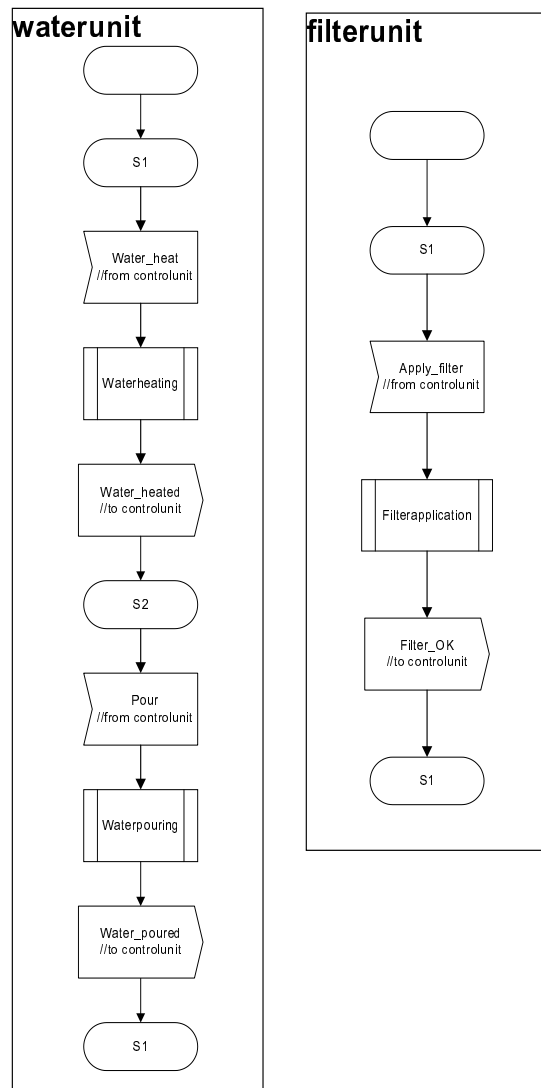


Figure 11: SDL process diagram. waterunit, filterunit

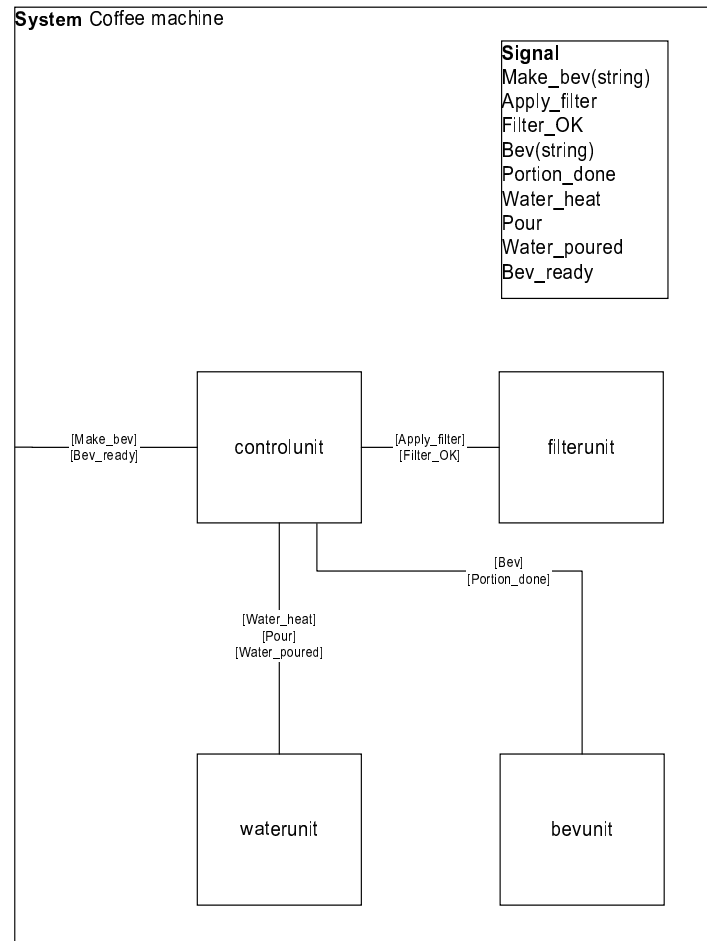


Figure 12: SDL block diagram