



# A language-mapping approach to action-oriented development of information systems

Peter Rittgen<sup>1</sup>

<sup>1</sup>*School of Business and Informatics, University College of Borås, Borås, Sweden*

**Correspondence:**

Peter Rittgen, School of Business and Informatics, University College of Borås, Allégatan 1, Borås 501 90, Sweden.  
Tel: +46 33 435 59 30;  
Fax: +46 33 435 40 07;  
E-mail: peter.rittgen@hb.se

**Abstract**

Two important views in the development of information systems are the action view and the reaction view which govern the areas of business process modelling and information systems modelling, respectively. We suggest an approach to mediate between these partially conflicting views by specifying a language-mapping framework. In other words, we specify a transition from a process model with human actors to an IS model with inanimate agents.

*European Journal of Information Systems* (2006) 15, 70–81.

doi:10.1057/palgrave.ejis.3000597

---

**Keywords:** language mapping; DEMO; UML; action view; reaction view

**Introduction**

In a re-engineering project it is not uncommon to encounter a situation where none of the available modelling methods can complete the whole task. We met such a situation in an interorganizational project where we used one such approach, the language-action perspective, to analyse the business process. This approach was useful for understanding the business situation, eliciting problems and proposing the design of an improved organization. But some of the problems consisted in poor or missing information system support and we considered that UML would be more useful for finding solutions to them. That left us with the task of ‘implementing’ our organizational designs, which we developed in DEMO, in a new language, UML, a task that was by no means straightforward owing to the fact that both languages represent different views on a system (the following paragraphs elaborate this point). As a consequence we developed a language-mapping framework to support this task. In the remaining sections we introduce the languages involved, develop the mapping procedure and relate empirical findings regarding this approach.

In the literature on business process modelling the (communicative) action view plays an important role (Winograd & Flores, 1986). This view ‘states that the major role of an information system is to support communication within an organization by structuring and coordinating the actions performed by the organization’s agents. The system is seen as a medium through which people can perform social actions, such as stating facts, making promises and giving orders’ (Johannesson, 1995, p. 291). Contrary to that, the modelling of information systems is often influenced by a view that we might term ‘reaction view’. According to this view, ‘the primary purpose of an information system is to provide a model of a Universe of Discourse (UoD), thereby enabling people to obtain information about reality by studying the model. In this respect, an information system works as a passive repository of data that reflects the structure and behaviour of the UoD’ (Johannesson, 1995, p. 291).

---

Received: 11 May 2005  
Revised: 17 December 2005  
Accepted: 3 January 2006

In the action view, a system consists of a number of agents (people or organizational units) who interact with each other by communicating. The basic unit of communication is a speech act (Austin, 1962; Searle, 1969). An action pair is the smallest sequence of actions that has an effect in the social world (e.g. establishing a commitment). It consists of two speech acts: an utterance and the response (e.g. a request and the promise). On the third level, the workflow loop (or action workflow, Medina-Mora *et al.*, 1992) describes a communicative pattern consisting of two consecutive conversations that aim at reaching an agreement about (1) the execution of an action, and (2) the result of that execution. The left side of Figure 1 shows three examples of workflow loops that consist of the conversations 1 and 6, 2 and 5 and 3 and 4, respectively. The conversations are executed in the order of the numbers, that is the workflow loops are embedded in each other. Higher levels can be defined such as contract and scenario but the first three are sufficient for the purpose of this paper.

In the reaction view, object orientation prevails today in many areas of software engineering. It has largely replaced the functional paradigm that characterized early approaches to software engineering (and is still used in certain areas such as databases). In object orientation, a system is seen as a collection of objects exchanging messages. Each object encapsulates data and functionality (or structure and behaviour or attributes and operations). An object is in principal a passive (or reactive) unit that only acts if it receives a message. It will then carry out the appropriate operation which might involve sending messages to other objects. Finally, it will deliver the result as a reply to the original message but 'communication' is essentially one-way (see Figure 1, right; the numbers refer to the temporal order of the messages).

The major conceptual differences between the views are:

1. The action view describes social systems that consist of human beings that can both act of their own accord and react to stimuli from the environment whereas an object can only react. This is why we have called that view 'reaction view'.

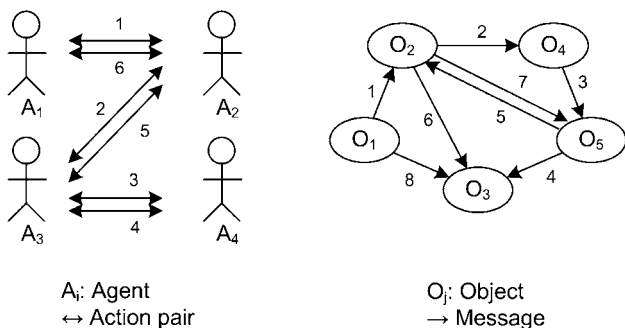


Figure 1 Action view and reaction view.

2. By performing speech acts, agents create obligations for themselves or others. Having a conscience they are fully aware of the consequences of entering into a commitment and also of not fulfilling an obligation. An object is not equipped with a conscience so it cannot commit itself. If an object behaves in the 'desired' way this is due to a pre-programmed automatism and not the result of an individual decision based on free will. An object cannot be responsible for its 'actions'.
3. Communicating is not just exchanging messages. We communicate to achieve a certain purpose for which we need the help of others. An object sends a message because its code prescribes this behaviour and the message is received, processed and 'answered' for precisely the same reason. An object has no intentions.

In short it can be said that the difference between agents and objects is the same as that between human beings and machines. In the light of these fundamental discrepancies it seems doubtful whether a reconciliation of action view and reaction view can be successful. But both views play an important role in the development of information systems as socio-technical systems so we cannot drop either of them. It would be equally inadequate to describe a social system in terms of mindless objects as it would be to describe a technical artefact such as software in terms of responsible and conscious agents. But a mere coexistence of both views is not enough because the social and technical aspects of an information system are so tightly interwoven that we cannot neatly separate them into weakly connected areas of concern, each dominated by its respective view. So we have to think of ways to 'mediate' between both views. Mediation is in our definition a weaker form of integration than reconciliation as defined in Johannesson (1995).

The approach we suggest operates on the language level. In the next section, we argue for this decision and present our language of choice for each view, DEMO and UML, respectively. Assuming the precedence of the action view over the reaction view we introduce a framework for mapping DEMO to UML. This involves both a mapping of the language concepts and a transformation of the diagram types.

**Languages for action view and reaction view**

Our approach to mediating between action view and reaction view proceeds at the language level. On the one hand this gives us a richer understanding of each view by examining the details of the concepts of a language representing this view. In addition, it also offers the chance to provide more comprehensive and constructive support for mediation. But on the other hand, each language will also introduce additional features that are not a necessary pre-requisite of the respective view. The results of such an approach should therefore be considered with care because they might not generalize to all

languages of the action and reaction views. In particular, the selected languages might not even be 'typical' representatives of their respective views.

Regarding the reaction view the task of finding an appropriate language is not difficult. The software engineering community has subjected itself to a rigorous standardization process that resulted in the Unified Modelling Language (UML). It follows the object-oriented paradigm and is widely used in the design of information systems. Adhering to the reaction view its focus is more on the technical part of the information systems than on the organizational (i.e. social) part but the proponents of UML claim that it can also be used for the latter. As evidence for this standpoint they mention use cases and business processes. For the former UML provides a specific language construct: Use Case Diagrams. The latter were originally supposed to be represented as Activity Diagrams (with swim lanes) but a language extension called Enterprise Collaboration Architecture (ECA; OMG, 2004) now takes care of business processes. Nevertheless, UML does not offer an action view because the concept of an actor is weakly integrated and restricted to the role of a user of the information system. This is rooted in the metamodel of the UML that does not provide concepts related to social action. Mechanisms that extend the language, such as stereotypes, cannot change the metamodel. They can therefore not extend UML in such a way that it would cover the action view. The role of the actor in both languages is argued more thoroughly in the section Concept mapping.

The situation regarding the action view (or language-action perspective) is much more complex. The approaches assuming this view cover a wide range of epistemological orientations coming from disciplines as diverse as social sciences, political science, law, linguistics, cognitive science, organizational theory, artificial intelligence and computer science. They have in common that they are based on Speech Act Theory (Austin, 1962; Searle, 1969) and the Theory of Communicative Action (Habermas, 1984). Examples of such approaches are Conversation-for-Action (Winograd & Flores, 1986), DiaLaw (Lodder & Herczog, 1995), Multi-Agent Systems (Dignum & Weigand, 1995; Hulstijn *et al.*, 2004), Dynamic Essential Modelling of Organizations (DEMO; DEMO-1: Dietz, 1999; DEMO-2: Liu *et al.*, 2003; Dietz & Habing, 2004), Action Workflow (Medina-Mora *et al.*, 1992; Denning & Medina-Mora, 1995; Kethers & Schoop, 2000), Action-Based Modelling (Lehtinen & Lyytinen, 1986), Business Action Theory and SIMM (Goldkuhl & Röstlinger, 1993; Goldkuhl, 1996; Goldkuhl & Lind, 2004) and Discourse Structures (Johannesson, 1995). As we aim at finding a language that is suitable for being mapped to UML we would like it to exhibit certain external similarities with UML: on the syntactic level it should provide a diagram-like notation and on the semantic level it should possess an appropriate degree of formality. We found that DEMO fulfils these criteria

best. The following sections give short introductions to both languages.

### Dynamic essential modelling of organization

In the action view, the structure of an organization is understood as a network of commitments. As these commitments are the result of communication, it follows that a model of the organization is essentially a model based on purposeful, communicative acts. In DEMO, all acts that serve the same purpose are collected in a *transaction* in which two roles are engaged: the *initiator* and the *executor*. The definition of a transaction in DEMO is broader than that given in Introduction. It comes closer to that of a workflow loop but it also includes a non-communicative action, namely the agreed action that the executor performs in the object world. Hence each transaction is assumed to follow a certain pattern which is divided into three sequential phases and three layers. The phases are: *order* (O), *execute* (E) and *result* (R). The layers are: success, discussion and discourse. On the success layer the phases are structured as follows. In the order phase the contract is negotiated. This involves typically a *request* being made by the initiator and a *promise* by the executor to carry out the request. In the next phase, the contract is executed which involves factual changes in the object world (as opposed to the intersubject world of communication). Finally, in the result phase the executor *states* that the agreed result has been achieved and the initiator *accepts* this *fact*. If anything goes wrong on the success layer, the participants can decide to move to the discussion or discourse layer. For details on these layers see (Reijswoud, 1996).

Figure 2 gives an overview of the architecture of DEMO. The Interaction Model shows actors and their relations to transactions but abstracts from time. The Business Process Model, on the other hand, abstracts from the actors but refines the transactional logic in two ways: it breaks each transaction into its phases and specifies how they are ordered causally and conditionally. This allows us to determine the order of the communicative acts in time. The Fact Model describes all information that is created

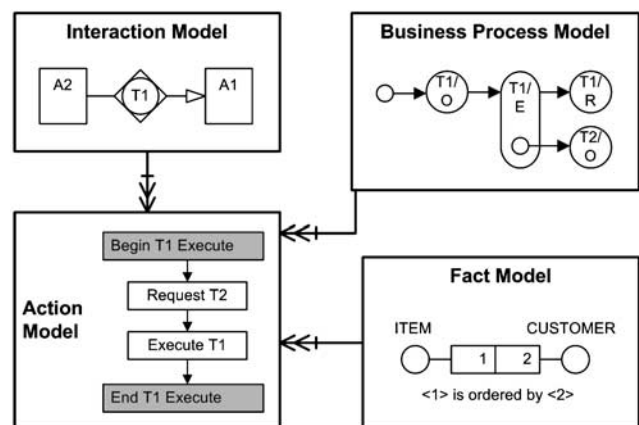


Figure 2 Architecture of DEMO (simplified).

or used by an organization. A fact is the result of a successful transaction and implies that the proposition of the request has become true. The Interstriction Model (not shown in Figure 2) is similar to the Interaction Model but in addition to the communication that is part of the transactions it also exhibits informative communication. All models are linked to the Action Model which gives a detailed account of activities carried out within a transaction phase (which can also involve links to other transactions).

Figure 3 gives examples of an Interaction Model and a Business Process Model. They are taken from (Reijswoud & Dietz, 1999) and show a part of the business process of an organization called SGC, a non-profit organization that mediates consumer complaints in the Netherlands.

The transactions of the example are as follows:

- T6: Handling\_complaint
- T7: Defending\_complaint
- T8: Giving\_advice
- T9: Passing\_judgement

The actor A2, who is responsible for mediating the claim, requests that actor A6 handles the complaint. Both A2 and A6 are internal actors (represented by white boxes). The latter will give the supplier a chance to defend the complaint, ask an expert to give advice and request that the committee passes a judgement. S2–S4 are external actors as the grey boxes show. A simple line connects the initiator with the transaction, an arrow points from it to the executor. The grey line represents the system boundary. Observe that Figure 3 shows only a fragment of the model.

The right side of Figure 3 contains a part of the Business Process Model. It shows details of the execution phase of transaction 6: Handling\_complaint. This phase is called T6/E. From inside it, the transactions T7, T8 and T9 are started. This is represented by arrows from the initiation points (small, white circles) to the order phases of the respective transactions. Solid arrows indicate a causal relation, dashed ones a conditional relation. The inside of a phase is viewed as a concurrent region, so all three triggered transactions could start at the same time if it were not for the dashed lines. An arrow that is crossed by a line represents an optional relation. In short: the supplier is asked to defend the complaint in any case

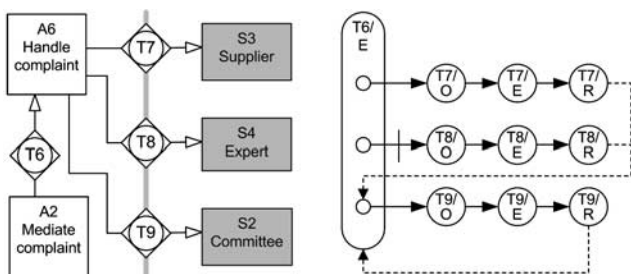


Figure 3 Examples of an Interaction Model and a Business Process Model.

and the expert is possibly consulted. After T7 (and possibly T8) have been completed, the committee is asked to pass judgement. Only when T9/R has been finished can we also terminate T6/E. Observe that this is due to the dashed arrow between them. As a rule the initiation points inside a phase trigger transactions in an asynchronous manner without waiting for their completion.

The UML

The specification of the UML (OMG, 2003) is divided into two parts: semantics and notation. The first part introduces the concepts of UML with the help of metamodels and natural language. It is organized in packages (which are themselves a concept of UML). See Figure 4 for an overview of the relevant packages.

All concepts for structural models are defined in the Foundation. They comprise static aspects of a system such as classes, interfaces and attributes. Based on the Foundation the elements of the behavioural (dynamic) models are specified which consist of Common Behaviour (such as signals, procedures, instances etc.) and diagram-specific behaviour (one package for each, see Figure 4). Their purpose is defined in OMG (2003, p. 2–92f.) as: ‘The Collaborations package specifies a behavioural context for using model elements to accomplish a particular task. The Use Case package specifies behaviour using actors and use cases. The State Machines package defines behaviour using finite-state transition systems. The Activity Graphs package defines a special case of a state machine that is used to model processes. The Actions package defines behaviour using a detailed model of computation.’

The notation part of the language specification introduces a number of diagrams that define how the elements of the semantics packages can be represented graphically. For the purpose of this paper the relevant diagrams (and the primary packages they refer to) are: Collaboration Diagram (Collaborations), Statechart Diagram (State Machines), Activity Diagram (Activity Graphs) and Class Diagram (Foundation). For a detailed description of these diagrams refer to OMG (2003).

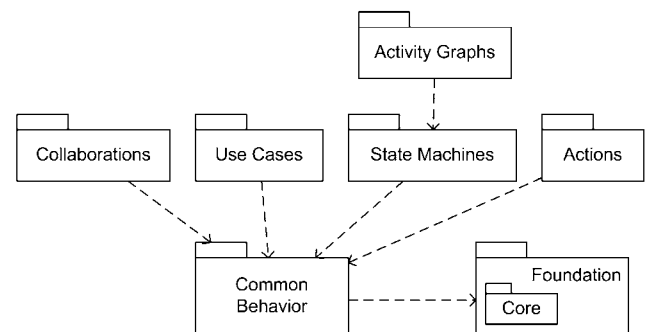


Figure 4 Architecture of UML (simplified).

## A language-mapping framework for DEMO and UML

### A general framework for mapping languages

The Object Management Group (OMG) has suggested an architecture for language integration that is called Model-Driven Architecture (MDA; Miller & Mukerji, 2003). In it a system is specified from three different viewpoints: computation independent, platform independent and platform specific. Although the scope of MDA is much broader, a typical assumption is that all models (views) can be constructed with the help of only one language, and UML is the preferred candidate for that role. But Evans *et al.* (2003) argue that 'a truly flexible model-driven development process should not dictate the language that practitioners should use to construct models, even an extensible one. Instead they should be free to use whichever language is appropriate for their particular domain and application, particularly as many languages cannot be shoe-horned into the UML family'. We follow this argument and suggest extending the model mapping of MDA (Caplat & Sourrouille, 2003) to 'language mapping'.

A general framework for mapping a modelling language to another one is shown in Figure 5. We distinguish between the conceptual level and the instance level. On the conceptual level, we first perform concept mapping. This step involves finding for each concept of the source language a matching one in the target language. A successful match implies that a significant part of the semantics of the source concept can also be expressed by the target concept. Note that concept mapping as defined here does not relate to the one known from (empirical) social research. For example, the DEMO concept of an action maps to the UML concept of an action state. The latter is something that is performed while the system is in a certain state. As this is very general, it encompasses the meaning of action in DEMO for which the same holds, but in addition to that an action is restricted to

being either an objective action in the object world or a communicative action in the intersubject world (see section Concept mapping for more details). Note that such a mapping is not always possible because the target language might not have a related concept at all or the 'common denominator' between both concepts is not a significant part of the semantics of the source concept (i.e. the two concepts have very little in common). This implies that language mapping cannot be done for any combination of languages, at least not in the way described here. Moreover, we cannot expect that we always succeed in establishing a one-to-one correspondence between concepts. Sometimes several source concepts jointly map to one target concept or one source concept maps to a conjunction of target concepts.

The second step consists of a notational mapping. We assume that each concept is associated with a notational element in the respective language so with concept mapping being done this step is straightforward. The third and last step is about establishing a relation between the diagram types of both languages. This step provides rules for carrying out the actual diagram transformation on the instance level. In the example of Figure 5 this step is trivial: it involves only a (slight) change in the notation and a rearrangement of the six basic concepts (three node types and three relation types: solid arc, dashed arc and arrow) into different diagrams. But for realistic modelling languages the mapping rules can be much more complex. Typical types of transformations include:

1. One element has to be mapped to a number of elements (e.g. a subgraph). This process is called unfolding.
2. Additional elements (of a different type) have to be introduced. This process is called element introduction.
3. Nodes are transformed into arcs (called node inversion) or arcs are transformed into nodes (arc inversion).
4. A substructure of the source language is transformed into a different substructure of the target language. This is called graph conversion.

In the following sections we specialize this framework for DEMO as the source language and UML as the target language.

### Concept mapping

Mapping concepts from one language to another requires an ontological analysis of both languages and a matching of corresponding concepts. The Bunge-Wand-Weber ontology (BWW ontology) is an established tool for analysing modelling languages. It is based on Mario Bunge's ontology (Bunge, 1977, 1979) and was later adapted by Yair Wand and Ron Weber to the information systems field (Wand & Weber, 1989, 1995; Weber, 1997). According to this ontology the world consists of *things* that possess *properties*. Both exist irrespective of the

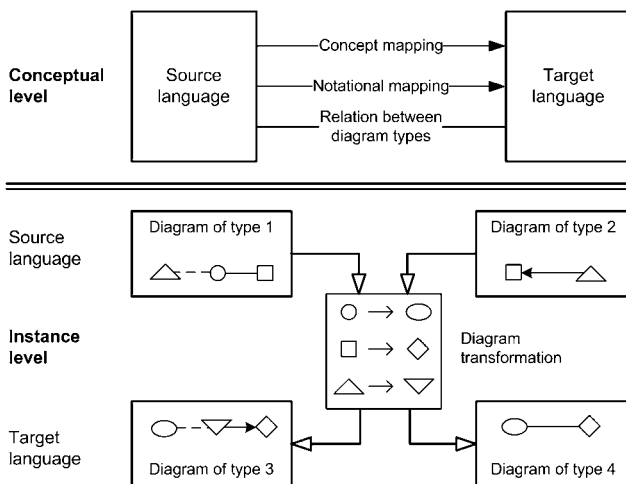


Figure 5 A framework for language mapping.

human observer. An *intrinsic property* belongs to an individual thing, a *mutual property* to two or more things. The observer can only witness *attributes* that he takes for the properties of the things he observes. The things themselves are not observable. The attributes are functions over time (*state functions*). A *class* is a set of things that share a property, a *kind* is a set of things sharing two or more properties and a *natural kind* is a kind where some of the properties are related by laws. A *law* is a relation between properties. A set of attributes used to describe a set of things with common properties is called a *functional schema*. The *state* of a thing is a complete assignment of values to all state functions in the functional schema. A change of state is called an *event*. A *lawful transformation* (or *transformation law*) defines which events in a thing that are lawful.

Let us look at five core concepts of DEMO, that is actor, action, transaction, phase and category, and how they can be mapped to UML concepts. An analysis of UML based on the BWW ontology has already been performed by Evermann & Wand (2001) and Opdahl & Henderson-Sellers (2002). The interpretation mapping of the respective UML constructs is shown in columns 2 and 3 in Table 1, that of the DEMO constructs in columns 1 and 3 of the same table. The resulting mappings from DEMO to UML are explained in the following sections.

**Actor → actor/object** The concept of an actor in DEMO is mapped both to that of an actor and to that of an object in UML. An actor (in DEMO) is a role that a subject (human being) plays. An actor (in UML) ‘defines a coherent set of roles that users of an entity can play when interacting with the entity. An actor may be considered to play a separate role with regard to each use case with which it communicates’ (OMG, 2003, p. 2–131). In UML an actor can play many roles, in DEMO it can play only one role (i.e. the actor is the role). But in UML actors can only interact with the information system. In DEMO they interact with each other. As use cases (or a similar notion) are not defined in DEMO, the mapping actor → actor is valid but not helpful. But if we also introduce a mapping actor → object we can at least interpret (or view) the actor as a (re)acting object. This means that we will lose the mind of the actor but we can still express a significant part of the semantics of an actor:

the structure and behaviour of the actor are retained in the object.

**Action → action [in an action state]** The concept of an action in DEMO is mapped to that of the same name in UML. In DEMO actions are divided into communicative actions and objective actions. Communicative actions take place in the intersubject world and consist of speech acts. Objective actions are performed in the object world and can be material or immaterial. Communicative actions can lead to changes in the social (intersubject) world (i.e. creating obligations), objective actions can lead to changes in the object world (i.e. the creation of facts). An action state refers to a state (of the system). Upon entering the state the (entry) action is triggered. Upon completion of the action the state is left. As the latter definition does not restrict the type of action it also encompasses the action concept of the former. In an Activity Diagram the action is associated with a state during which the action is executed (action state).

**Transaction → sequence of messages (operations) [in a state]** The concept of a transaction in DEMO is mapped to that of sequence of messages in UML where each message triggers a corresponding operation. A transaction is a pattern that consists of the conversation for an (objective) action, the execution of that action and the conversation for the result (of that action). This concept has no direct counterpart in UML. Instead it has to be mapped to a sequence of messages which represent the speech acts. Technically this is done by unfolding on the instance level (i.e. transformation interaction model → collaboration diagram). In the Statechart Diagram a transaction is represented by a state during which the messages are passed. When the system enters this state the transaction is initiated. When the transaction is finished the system leaves the state.

**Category → class** The concept of a category in DEMO is mapped to that of a class in UML. A category is a primal class (derived classes are used to define roles in DEMO). Each object is an instance of exactly one category but can also be an instance of arbitrarily many derived classes. In UML ‘a class is a description of a set of objects that share the same attributes, operations, methods, relationships and semantics’ (OMG, 2003, p. 2–26). A category (as a

Table 1 Interpretation mapping of DEMO and UML constructs

DEMO construct	UML construct	BWW construct
DEMO-actor	UML-actor/UML-object	BWW-thing (that acts on the proposed system thing)
DEMO-action	UML-action (in an action state)	BWW-transformation law that describes a single event
DEMO-transaction	UML-sequence of messages (operations) (in a state)	BWW-transformation law that describes a process
DEMO-phase	UML-sub state	BWW-state
DEMO-category	UML-class	BWW-(functional schema for modelling entities that form the corresponding) natural kind

primal class) is a class in the sense of this definition. Categories can have  $n$ -ary relations which correspond to association classes in UML. Categories and their relations are used to represent facts about the intersubject world and the object world.

### Diagram transformation

Figure 6 shows the overall framework for the language mapping. The DEMO diagrams are represented by rounded boxes, the UML diagrams by rectangular boxes. The mapping of concepts is visualized by single-headed arrows, the transformation of diagrams by double-headed arrows. Each diagram conversion involves a transformation of the notation but will also require some more sophisticated transformation process (e.g. transaction unfolding). Concurrency explication and class association are graph conversions, Signal and Infolink introduction are element introductions, and Transaction unfolding is an unfolding in the sense of the general integration framework.

The Interaction Model introduces actors and transactions. The actors become objects in the Collaboration Diagram, the transactions are transformed into sequences of messages (operations) in the same diagram but they correspond also to states in the Statechart Diagram. The Business Process Model refines transactions into phases which in turn become substates of the respective transaction state in the Statechart Diagram. The basic elements of the Fact Model are the categories. They correspond to classes in UML. The Interstriction Model introduces fact and communication banks to store records of facts and communication. They also correspond to classes in UML. Actions and wait states in the

Action Model become actions (in action states) and signal receipts, respectively, in the Activity Diagram.

The Interaction Model is transformed into the Collaboration Diagram. Apart from a notational conversion this requires an unfolding of the transactions, a concept which has no immediate dynamic counterpart in UML. Each transaction is split into its communicative acts which then are represented by messages in UML. An example of that is given in the next section.

The Business Process Model is transformed into the Statechart Diagram. Again this involves a change in notation but also an explication of the inherent concurrent behaviour of a phase. A phase can have many concurrent initiation points but each state has only one initial (sub)state. Dividing the state into concurrent regions is not feasible due to the asynchronous nature of the threads triggered by the initiation points. Hence the initial state is forked into as many threads as there are initiation points that have no arrows pointing at them (plus one that leads to the final state if no arrow points to the phase). An arrow pointing at a phase maps to one pointing at the corresponding final state. If more than one arrow points at a phase or initiation point the respective arrows in the Statechart Diagram are joined by a synchronization bar. Optional relationships map to guarded transitions. An example for such a transformation is given in the next section.

The Action Model is transformed into the Activity Diagram. Apart from the usual notational conversion this means that a signal receipt has to be introduced into the Activity Diagram for each wait state that is found in the Action Model. Likewise, a signal sending is introduced after the activity that corresponds to the action that is waited for.

The Fact Model is transformed into the Class Diagram. This involves that each fact (which is an  $n$ -ary relation between categories) is mapped to an association class that has associations to each of the classes corresponding to the categories. That process is called class association.

The Interstriction Model introduces further associations into the Class Diagram, one for each informational link between an actor and a transaction, fact bank or communication bank. We call that process infolink introduction.

### Examples of diagram transformation

Due to the limited space we give examples for the first two transformations only. Figure 7 shows the Collaboration Diagram (upper half) for the Interaction Model of Figure 3 (left) and also the Statechart Diagram (lower half) for the Business Process Model of Figure 3 (right). Each system or actor of the Interaction Model becomes an object (instance) in the Collaboration Diagram. A transaction is represented by a (communication) link that bears the name of the transaction (i.e. its purpose). This link is bidirectional (i.e. it does not have an arrowhead that restricts the navigability) because a transaction involves communication in both directions, from

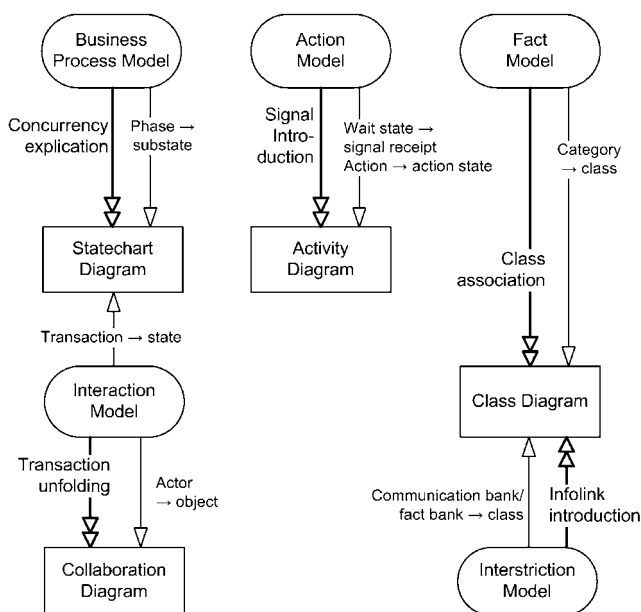


Figure 6 Framework for integrating DEMO and UML.

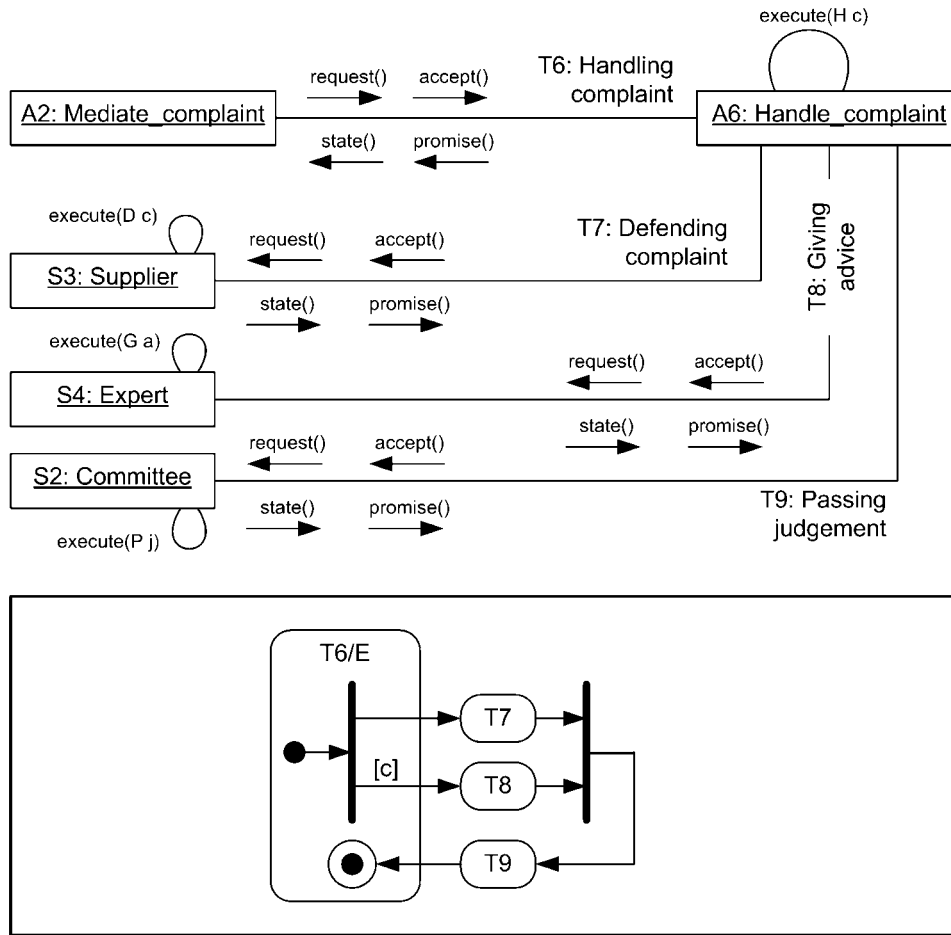


Figure 7 Collaboration Diagram and Statechart Diagram.

initiator to executor and back. This link can now be used to exchange the messages that correspond to the communicative acts in DEMO. Each executor has also a link to itself which means that the execution phase is self-induced. A request and an accept message are introduced along the link with arrows that point from the initiator to the executor. They represent the first and the last communicative acts of a transaction, respectively. In the same way, a promise and a state message are attached to the link. They are passed from the executor to the initiator and form the second and penultimate speech acts, respectively. Observe that a Collaboration Diagram does not require us to specify the order of messages but we could do so with the help of sequence numbers in front of the message names.

The lower half of Figure 7 shows the Statechart Diagram that corresponds to the excerpt from the Business Process Model of Figure 3. The execution phase of T6 becomes a state (which itself is a substate of the transaction state T6). Within T6/E the initial state is forked into two concurrent threads to trigger transactions T7: Defending\_complaint and T8: Giving\_advice. While T7 is triggered in any case, the transition to T8 is guarded

by [c], which means that the expert is asked to give advice under a condition that has not yet been specified; the Business Process Model only indicates that T8 is optional, not under which circumstances it is carried out. On completion of T7 (and possibly T8), T9: Passing\_judgement is carried out. After that we enter the terminal state of T6/E which concludes the execution phase of T6.

**Empirical studies**

**An interorganizational case study**

The ideas in this paper were inspired by a project we carried out in spring 2004 together with two companies: a logistics provider and a large retail chain. The objective was to model the complex interorganizational business process as a basis for its reorganization. We found that the language-action perspective was successful in that scenario. One of the reasons for this is certainly the highly interactive nature of the process we studied where communication is vital and frequent. But LAP also facilitated understanding among people who not only came from different organizations but also worked in different domains: purchase, marketing, inbound and



outbound logistics etc. It made a complex process more transparent to all participants (each of whom provided only a small puzzle piece to the overall picture) and it allowed them to discuss in a constructive way possible options of reorganization. As a result two major areas for improvement were identified: a tighter integration between the different information systems of both companies and a greater accuracy in the forecasts concerning incoming and outgoing commodity flows.

The framework that we have presented helped us in developing an approach to solve the first problem. The study proceeded in three steps. In the first step we analysed from a language-action perspective those parts of the two businesses that require cooperation. As a result we created detailed models of information flows and information dependencies, problem graphs and goal graphs together with a comprehensive, textual description of the businesses and their problems in relation to their cooperation. Figure 8 shows a part of the data flows for Receiving goods.

Both companies operate their own warehouse management system. The physical warehouse at the logistics

provider is managed with DISA. During the night a file detailing the goods received during the day is sent to the retail chain to update the ‘virtual’ warehouse which is managed with SAP. This ‘double bookkeeping’ often leads to inconsistencies between the two databases, for example, if a file is not received in order.

In the second step we decided to address one of the problems that were elicited in the first step, namely that of integrating the respective information systems. We did so by developing Interaction Models to bring the communicative structure to the surface. A simple example is shown in Figure 9 (top) concerning the inconsistency issue. We thereby identified the respective warehouse managers as the actors who are responsible to negotiate and control the synchronization of the warehouses.

In the third step we used language mapping to create initial UML models for the design of an appropriate information system support. Figure 9 (bottom) shows the part of the Collaboration Diagram for transaction T2: Register goods. The phases ‘promise’ and ‘accept’ can be omitted as they are provided for in a framework contract. In the light of this model we were able to assess that the ‘state’ phase is not implemented in the current system. As a consequence, a failure in receiving the update file cannot be recognized by the sender and the file is not resent.

**The language-mapping experiment**

In order to check the plausibility of the language-mapping described in the third section and to get additional insights we carried out an experiment. We provided eight master students in their final year with four DEMO models of a simple order processing. From January 28 to February 4, 2005, they were supposed to express all information contained in the DEMO models with the help of an arbitrary selection of UML diagrams without any knowledge of the mapping framework. The resulting models were to be complemented by a questionnaire consisting of three parts: a table specifying the

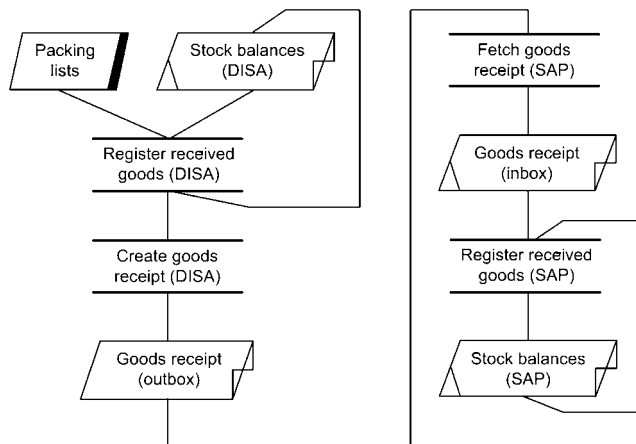


Figure 8 Data-flow diagram of Receiving goods (excerpt).

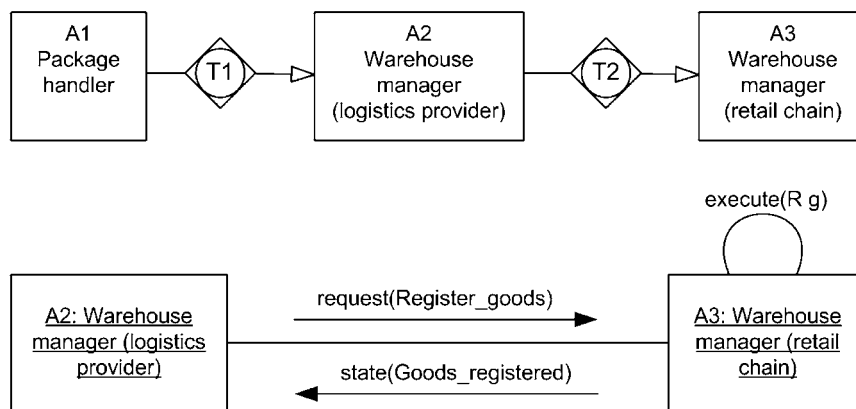


Figure 9 Partial Interaction Model (top) and Collaboration Diagram (bottom).

concept mapping, a graph detailing the relations between DEMO and UML diagrams, and some general questions regarding the mapping process. The results of concept and diagram mappings are shown in Tables 2 and 3.

Concerning Fact Diagram and Business Process Diagram the empirical results coincide completely with the proposed language mapping. The Action Diagram has been translated into an Activity Diagram by most participants, whereas our conceptual approach suggested a Statechart Diagram here. But this can also be counted as an agreement because the Activity Diagram is a special case of a Statechart Diagram.

The case of the Interaction Diagram is more involved. The participants of the experiment considered the Use Case Diagram as the most appropriate counterpart. The answers to part 3 of the questionnaire suggest that this choice was motivated by the fact that Use Case Diagrams are the only ones in UML that make explicit reference to actors. But empirical studies such as (Maij *et al.*, 2002) found that the mapping from DEMO to Use Cases is not straightforward. But nevertheless this indicates that it is worthwhile to extend the framework to Use Cases.

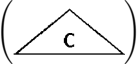
### Related work

The work described in this paper is closely related to that of Ågerfalk & Eriksson (2004). They share our goal to develop conceptual models of the information system from action-oriented business process models. On the surface the actual approaches to reach this goal seem to

have very little in common but many of the dissimilarities are of a minor nature. For example, Ågerfalk & Eriksson (2004) use SIMM instead of DEMO to represent the action view. Only two major differences can be identified: Ågerfalk & Eriksson (2004) focus primarily on static target models (Class Diagrams or Entity-Relationship Diagrams) with the ultimate aim of database design whereas our approach targets both static and dynamic UML models. But on the other hand do we require that a complete analysis of the business process in DEMO has been done, which includes the Fact Diagrams that we need to derive Class Diagrams. But not every business analysis includes Fact Diagrams and in such a situation Ågerfalk & Eriksson (2004), which require only Action Diagrams, can be a valuable complement to the language-mapping approach.

Our work can also be seen as a part of situational method engineering. Method engineering as such has been introduced by Harmsen *et al.* (1994) and has its roots in methodology engineering (Kumar & Welke, 1992). Method engineering has been studied in the context of the action view (Ågerfalk & Fitzgerald, 2005). The principal idea behind situational method engineering (Brinkkemper *et al.*, 1999) is to design a method in such a way that it fits the requirements of a particular situation (i.e. a project) as closely as possible. This can be done in different ways (Ralyté *et al.*, 2003). One way is to extend an existing method. Another one is to create a new one from chunks of existing methods by performing method chunks selection and assembly. The third way is to construct a new method from scratch with the help of a suitable metamodel or paradigm. One strategy for method chunks assembly is called association strategy that is used when the chunks correspond to two different

**Table 2 Concept mapping (summary)**

<i>DEMO concept</i>	<i>UML concept</i>
Actor	<i>Actor</i> , role, swim lane
System	<i>System</i> , swim lane
Transaction	Use case, activity state, <i>activity</i>
Transaction phase (O, E, R)	Signal, state, operation, actor, set of activities
Communicative action	<i>Actor-use case association</i> , activity
Objective action	<i>Activity</i> , state
State	Guard, initial state, <i>state</i> , activity, initial & final state
Wait for state (Action Diagram)	Activity state, synchronisation, <i>state</i> , activity
Selection 	Decision diamond
Causal relation	Decision diamond, <i>transition</i> , (no counterpart)
Optional causal relation	<i>Decision diamond</i> , guard, (no counterpart)
Conditional relation	<i>Decision diamond</i> , (no counterpart)
Initiation disk	<i>Initial state</i> , transition
Category	Object, <i>class</i>
Fact type (unary, binary, ternary)	Class, association, qualifier, (no counterpart)
Domain	<i>Association</i> , object

**Table 3 Diagram mapping**

<i>DEMO diagram</i>	<i>UML diagram(s)</i>
Fact Diagram	<i>Class Diagram</i> (4) Object Diagram (1) Both (1) No mapping (2)
Interaction Diagram	<i>Use Case Diagram</i> (5) Collaboration Diagram+Sequence Diagram (1) Use Case Diagram+Sequence Diagram (1) Use Case Diagram+Sequence Diagram (1)
Business Process Diagram	<i>Activity Diagram</i> (5) Use Case Diagram (1) Both (1) Sequence Diagram (1)
Action Diagram	<i>Activity Diagram</i> (6) Sequence Diagram (1) Statechart Diagram (1)

system engineering functionalities. Our work can be seen as an instantiation of this strategy where the languages, DEMO and UML, represent different chunks. The aim of this strategy is to make a bridge between the chunks, which coincides with the content of this paper.

## Conclusion

The purpose of this paper is to suggest one way of mediating between the action and reaction views by providing a mapping between their associated languages, DEMO and UML. This is done by mapping their respective concepts and eventually transforming diagrams of the former into corresponding ones of the latter. These two languages represent completely different paradigms: DEMO is an approach that is rooted in human communication and social action, while UML has its main roots in computer science. Therefore the transition from DEMO to UML cannot be performed without loss. That part of action that is connected with human agency and the social context is not preserved. A message cannot persuade an object to do something, nor can it appeal to an object's conscience. An object cannot commit itself

and it cannot be held responsible if it does not fulfil a commitment. Social rules have no meaning in the world of objects. But there is also another type of loss that is associated with the way that communication is structured. DEMO distinguishes different layers of actions, transactions, phases and speech acts that have no direct counterpart in UML. As a consequence language actions have to be unfolded into flat message sequences. This procedure is not reversible. An inverse mapping from UML to DEMO, although it is technically possible, cannot recover the original layered structure of the DEMO actions.

Given this incompatibility of the paradigms it is not self-evident that a mapping between these languages can be accomplished at all. It should be noted, though, that we have chosen from the set of all language-action approaches the one that best facilitates an integration with UML. Other languages of the action view might prove less suitable. Nevertheless, we hope that our work can contribute to narrowing the gap between organizational modelling and the design of information systems.

## About the authors

**Peter Rittgen** received a M.Sc. in Computer Science and Computational Linguistics from University Koblenz-Landau, Germany, and a Ph.D. in Economics and Business Administration from Frankfurt University, Germany. He is currently a senior lecturer at the School of Business and

Informatics of the University College of Borås, Sweden. He has been doing research on business processes and the development of information systems since 1997 and published many articles in these areas. For further details the reader is referred to <http://www.adm.hb.se/~PRI/>.

## References

- ÅGERFALK PJ and ERIKSSON O (2004) Action-oriented conceptual modelling. *European Journal of Information Systems* **13**(1), 80–92.
- ÅGERFALK PJ and FITZGERALD B (2005) Methods as action knowledge: exploring the concept of method rationale in method construction, tailoring and use. In *Proceedings of EMMSAD'05: Tenth IFIP WG8.1 International Workshop on Exploring Modeling Methods in Systems Analysis and Design* (HALPIN T, KROGSTIE J and SIAU K, Eds), pp 413–426, Porto, Portugal.
- AUSTIN JL (1962) *How to Do Things with Words*. Oxford University Press, Oxford, UK.
- BRINKKEMPER S, SAEKI M and HARMSSEN F (1999) Meta-modelling based assembly techniques for situational method engineering. *Information Systems* **24**(3), 209–228.
- BUNGE M (1977) *Ontology I: the Furniture of the World*. Holland, Dordrecht.
- BUNGE M (1979) *Ontology II: a World of Systems*. Holland, Dordrecht.
- CAPLAT G and SOURROUILLE JL (2003) Considerations about model mapping. In *Proceedings of the Workshop in Software Model Engineering WiSME@UML'2003* (BEZVIN J and GOGOLLA M, Eds), Online version available at <http://www.metamodel.com/wisme-2003/18.pdf>.
- DENNING PJ and MEDINA-MORA R (1995) Completing the loops. *Interfaces* **25**(3), 42–57.
- DIETZ JLG (1999) Understanding and modelling business processes with DEMO. In *Proceedings of the 18th International Conference on Conceptual Modelling ER'99* (AKOKA J, BOUZEGHOUB M, COMYN-WATTIAU I and MÉTAIS E, Eds), pp 188–202, Springer, Berlin, Germany.
- DIETZ JLG and HABING N (2004) The notion of business process revisited. In *Proceedings of the OTM Confederated International Conferences, CoopIS, DOA, and ODBASE* (MEERSMAN R and TARI Z, Eds), pp 85–100, Springer, Berlin, Germany.
- DIGNUM F and WEIGAND H (1995) Modelling communication between cooperative systems. In *Proceedings of the Seventh International Conference on Advanced Information Systems Engineering CaiSE'95* (IIVARI J, LYTYINEN K and ROSSI M, Eds), pp 140–153, Springer, Berlin, Germany.
- EVANS A, MASKERI G, SAMMUT P and WILLANS JS (2003) Building families of languages for model-driven system development. In *Proceedings of the Workshop in Software Model Engineering WiSME@UML'2003* (BEZVIN J and GOGOLLA M, Eds), Online version available at <http://www.metamodel.com/wisme-2003/06.pdf>.
- EVERMANN J and WAND Y (2001) Towards ontologically based semantics for UML constructs. In *Conceptual Modelling – ER 2001* (KUNII HS, JAJODIA S and SØLVBERGA, Eds), 20th International Conference on Conceptual Modelling, Yokohama, Japan, November 27–30, 2001, pp 354–367, Springer, Berlin, Germany.
- GOLDKUHL G (1996) Generic business frameworks and action modelling. In *Proceedings of the First International Workshop on Communication Modelling* (DIGNUM F, DIETZ J, VERHAREN E and WEIGAND H, Eds), Electronic Workshops in Computing, Springer, Berlin, Germany.
- GOLDKUHL G and LIND M (2004) The generics of business interaction – emphasizing dynamic features through the BAT model. In *Proceedings of the Ninth International Working Conference on the Language-Action Perspective on Communication Modelling LAP 2004* (AAKHUS M and LIND M, Eds), pp. 1–26, Rutgers University, New Brunswick, NJ, USA.
- GOLDKUHL G and RÖSTLINGER A (1993) Joint elicitation of problems: an important aspect of change analysis. In *Human, Organizational, and Social Dimensions of Information Systems Development* (AVISON D,

- KENDALL J and DEGROSS J, (Eds), North-Holland, Amsterdam, The Netherlands.
- HABERMAS J (1984) *The Theory of Communicative Action 1, Reason and the Rationalization of Society*. Beacon Press, Boston, MA, USA.
- HARMSSEN F, BRINKKEMPER S and OEI H (1994) Situational method engineering for information system project approaches. In *Methods and Associated Tools for the Information Systems life cycle. Proceedings of the IFZP WG8.1 Working Conference CRIS'94, Maastricht, September 1994*, (VERRIJN STUART AA and OLLE TW, Eds), pp 169–194, North-Holland, Amsterdam.
- HULSTIJN J, DIGNUM F and DASTANI M (2004) Coherence constraints for agent interaction. In *Proceedings of the Workshop on Agent Communication AAMAS 2004* (EIJK RMV, HUGET MP and DIGNUM F, Eds), pp 134–152, Springer, Berlin, Germany.
- JOHANNESSON P (1995) Representation and communication: a speech act based approach to information systems design. *Information Systems* **20**(4), 291–303.
- KETHERS S and SCHOOP M (2000) Reassessment of the action workflow approach: empirical results. In *Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling LAP 2000* (SCHOOP M and QUIX C, Eds), pp 151–169, RWTH Aachen University, Germany.
- KUMAR K and WELKE RJ (1992) Methodology engineering: a proposal for situation-specific methodology construction. In *Challenges and Strategies for Research in Systems Development* (COTTERMAN WW and SENN JA, Eds), John Wiley, New York.
- LEHTINEN E and LYYTINEN K (1986) An action based model of information systems. *Information Systems* **11**(4), 299–317.
- LIU K, SUN L, BARJIS J and DIETZ JLG (2003) Modelling dynamic behaviour of business organisations – extension of DEMO from a semiotic perspective. *Knowledge-Based Systems* **16**(2), 101–111.
- LODDER AR and HERCZOG A (1995) Dialaw – a dialogical framework for modelling legal reasoning. In *Proceedings of the Fifth International Conference on Artificial Intelligence and Law*, pp 146–155, ACM, New York, NY, USA.
- MAIJ E, TOUSSAINT PJ, KALSHOVEN M, POERSCHKE M and ZWETSLOOT-SCHONK JHM (2002) Use cases and DEMO: aligning functional features of ICT-infrastructure to business processes. *International Journal of Medical Informatics* **65**(3), 179–191.
- MEDINA-MORA R, WINOGRAD T, FLORES R and FLORES F (1992) The action workflow approach to workflow management technology. In *Proceedings of the Conference on Computer-Supported Cooperative Work CSCW'92* (TURNER J and KRAUT R, Eds), pp 281–288, ACM, New York, NY, USA.
- MILLER J and MUKERJI J (2003) MDA Guide Version 1.0.1. OMG, Needham, MA, USA. Online version available at <http://www.omg.org/docs/omg/03-06-01.pdf>.
- OMG (2003) Unified Modelling Language Specification: Version 1.5. OMG, Needham, MA, USA. Online version available at <http://www.omg.org/docs/formal/03-03-01.pdf>.
- OMG (2004) Enterprise Collaboration Architecture Specification. Version 1.0. OMG, Needham, MA, USA. Online version available at <http://www.uml.org>.
- OPDAHL AL and HENDERSON-SELLERS B (2002) Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Software and Systems Modelling* **1**(1), 43–67.
- RALYTÉ J, DENECKÈRE R and ROLLAND C (2003) Towards a generic model for situational method engineering. In *Proceedings of 15th International Conference on Advanced Information Systems Engineering (Caise 2003), Klagenfurt, Austria, June 16–18, 2003* (EDER J, MISSIKOFF M, Eds), pp 95–110, Springer-Verlag, Heidelberg, Germany.
- REIJSWOUD VEv (1996) The structure of business communication: theory, model and application. PhD Thesis, TU Delft, The Netherlands.
- REIJSWOUD VEv and DIETZ JLG (1999) DEMO Modelling Handbook. TU Delft, The Netherlands. Online version available at <http://www.demo.nl/documents/handbook.pdf>.
- SEARLE JR (1969) *Speech Acts, an Essay in the Philosophy of Language*. Cambridge University Press, London, UK.
- WAND Y and WEBER R (1989) An ontological evaluation of systems analysis and design methods. In *Information Systems Concepts: An In-Depth Analysis* (FALKENBERG ED and LINDGREEN P, Eds), pp 79–107, North-Holland: Amsterdam, The Netherlands.
- WAND Y and WEBER R (1995) On the deep structure of information systems. *Information Systems Journal* **5**, 203–223.
- WEBER R (1997) *Ontological Foundations of Information Systems*. Coopers & Lybrand and the Accounting Association of Australia and New Zealand, Melbourne, Australia.
- WINOGRAD T and FLORES F (1986) *Understanding Computers and Cognition: a New Foundation for Design*. Ablex, Norwood, NJ, USA.