Chapter XIV
From Business Process Model to Information Systems Model: Integrating DEMO and UML

Peter Rittgen
University College of Borås, Sweden

ABSTRACT

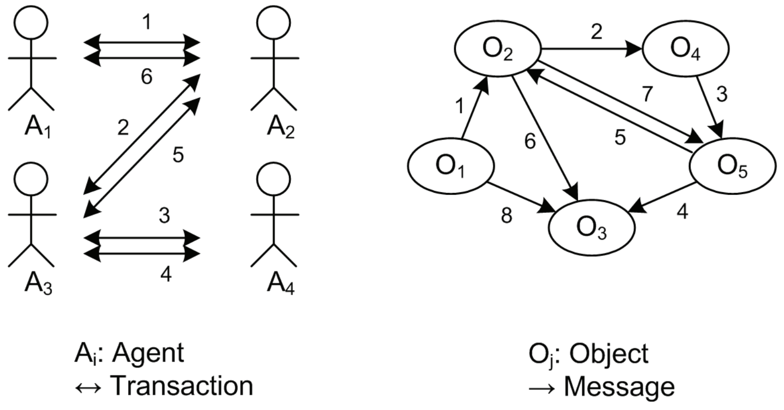
The main purpose of a corporate information system is the support of the company's business processes. The development of information systems is therefore typically preceded by an analysis of the business processes it is supposed to support. The tasks of analysing business processes and designing information systems are governed by two seemingly incompatible perspectives related to the interaction between human actors or inanimate agents (objects), respectively. As a consequence, the corresponding modeling languages also differ. DEMO (dynamic essential modeling of organization) is a typical language for modeling business processes, the UML is the predominant language for information systems modeling. We challenge the assumption of incompatibility of the perspectives by providing a framework for the integration of these languages.

INTRODUCTION

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). A transaction (Weigand & van den Heuvel, 1998) is the smallest sequence of actions that has an effect in the social world (e.g., establishing a commitment). It typically 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, Winograd, Flores, & Flores, 1992) describes a communicative pattern consisting of two consecutive transactions 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. Higher levels can be defined such as contract and scenario but the first three are sufficient for the purpose of this chapter. More details on the action view are given in the section "Dynamic Essential Modeling of Organization."

In the reaction view, object orientation prevails today. It has largely replaced the functional paradigm that characterized early approaches to software en-

Figure 1. Action view and reaction view



gineering (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). More details on the reaction view can be found in the object-oriented literature, for example (Dori, 2002).

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.
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.

BACKGROUND

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 modeling 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 swimlanes), 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. Moreover, empirical research shows (Dobing & Parsons, 2006) that UML is often not applied in a use case-driven way, which suggests a need for a different front-end that provides better communication with users.

The situation regarding the action view is 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 the speech-act theory (Austin, 1962; Searle, 1969; Habermas, 1984). Examples of such approaches are conversation-for-action (Winograd & Flores, 1986), DiaLaw (Lodder & Herczog, 1995), illocutionary logic (Dignum & Weigand 1995), dynamic essential modeling of organizations (DEMO, Dietz, 1999), action workflow (Medina-Mora, Winograd, Flores, & Flores, 1992; Denning & Medina-Mora, 1995), action-based modeling (Lehtinen & Lyytinen, 1986), business action theory & SIMM (Goldkuhl & Röstlinger, 1993; Goldkuhl, 1996), 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 dynamic essential modeling of organization (DEMO) fulfils these criteria best.

DYNAMIC ESSENTIAL MODELING 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 the 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 inter-subject 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). The models of DEMO are: interaction model, business process model, action model, fact model, and interstriction model. The strengths of DEMO lie in its systematic and stringent ontology that provides strong guidelines for modeling and fully implements the (communicative) action perspective. On the other hand, DEMO subscribes to a particular conceptualization of the world and does not allow expressing a system from another perspective, for example, the reaction view.

It therefore needs to be complemented by another language, in our case, UML.

Figure 2 gives examples of an interaction model and a business process model. They are taken from Reijswoud and 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:

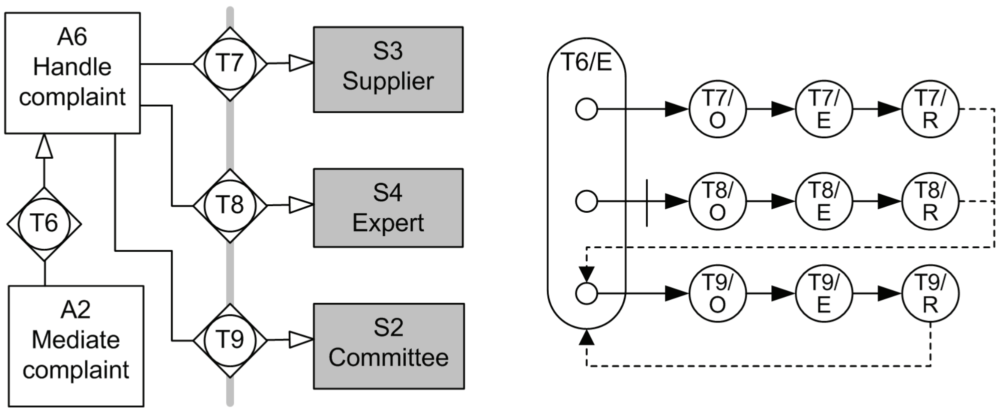
- T6: handling_complaint
- T7: defending_complaint
- T8: giving_advice
- T9: passing_judgement

A LANGUAGE-MAPPING FRAMEWORK FOR DEMO AND UML

The object management group (OMG) has suggested 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, Maskeri, Sammut, and Willans (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."

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

Figure 2. Examples of an interaction model and a business process model



either an objective action in the object world or a communicative action in the inter-subject world. 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. Typical types of transformations include:

1. One element has to be mapped to a number of elements (e.g., a sub graph). 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 subsections, we specialize this framework for DEMO as the source language and UML as the target language.

DIAGRAM TRANSFORMATION

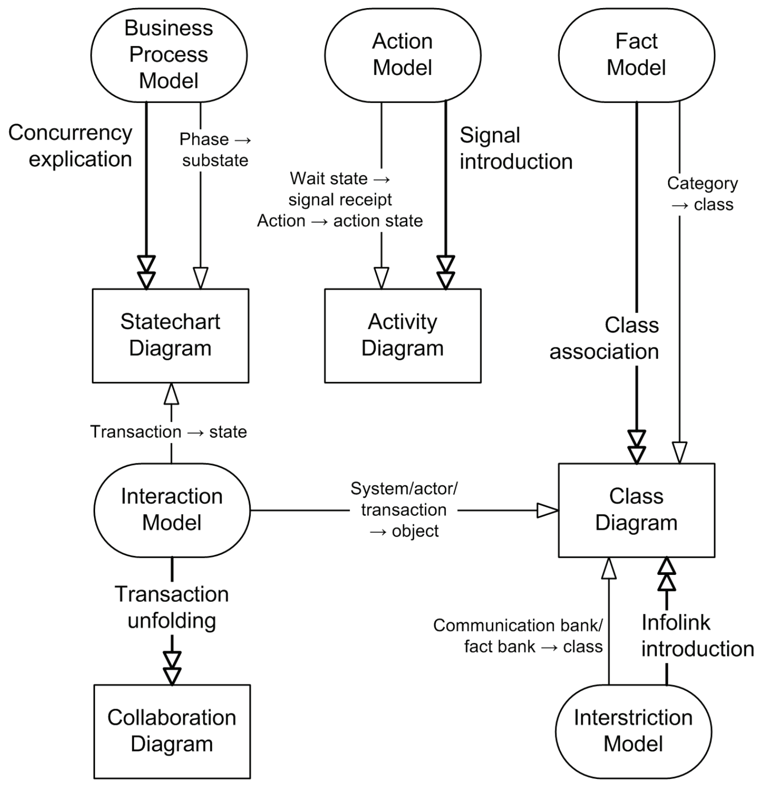
Figure 3 shows the overall framework for the language mapping. The DEMO diagrams are represented by rounded boxes, the UML diagrams by rectangular boxes. The mapping of concepts is visualized by single-headed arrows, the transformation of diagrams by double-headed arrows. Each diagram conversion involves 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 systems, actors, and transactions that all become classes in UML. But the transactions (the most important concept of DEMO) also form states in the statechart diagram. The business process model refines transactions into phases which in turn become sub states 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. The action model introduces wait states which map to signal receipts in activity diagrams.

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

Figure 3. Framework for integrating DEMO and UML



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 relations 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.

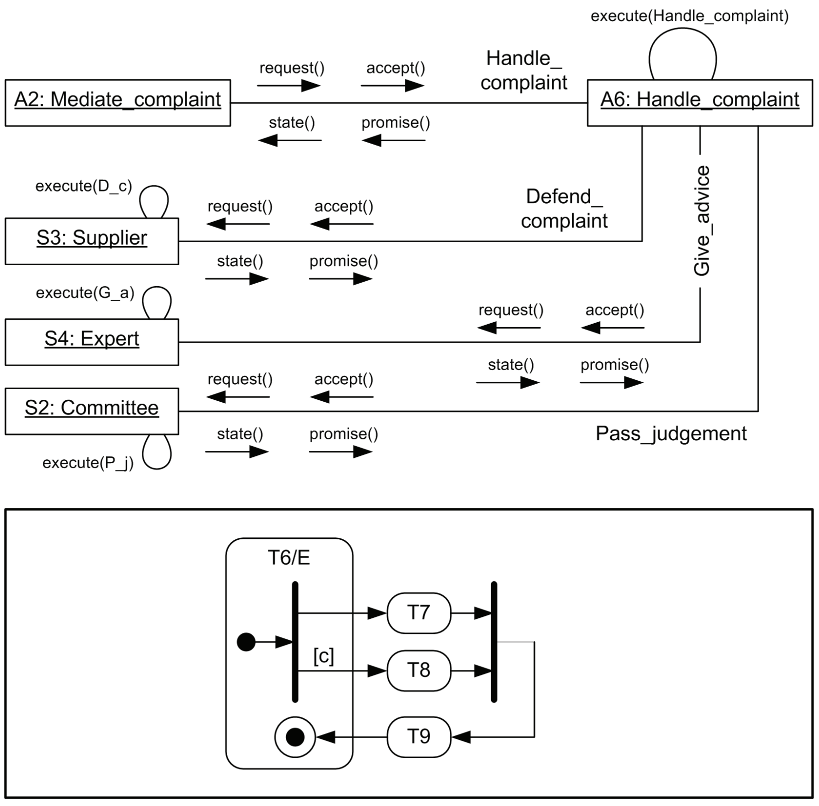
The interstriction model introduces further associations into the class diagram, one for each informational link between an actor and a transaction, fact bank, or communication bank. We call that process infolink introduction.

EXAMPLES OF DIAGRAM TRANSFORMATION

Due to the limited space, we give examples for the first two transformations only. Figure 4 shows the collaboration diagram (upper half) for the interaction model of Figure 2 (left) and also the statechart diagram (lower half) for the business process model of Figure 2 (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 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 4 shows the statechart diagram that corresponds to the excerpt from the business process model of Figure 2. The execution

Figure 4. Collaboration diagram and statechart diagram



phase of T6 becomes a state (which itself is a sub state 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_judgment is carried out. After that, we enter the terminal state of T6/E which concludes the execution phase of T6.

This and the preceding section give only an overview of the overall process. The details can be found in Rittgen (2006). The benefits of the language mapping include a smooth transition from the analysis to the design phase and an improved user-oriented front-end for UML.

FUTURE TRENDS

The work we have described can be seen as an implementation of a specific strategy of method engineering (Ralyté, Deneckère, & Rolland, 2003) called method chunks assembly, with the chunks being DEMO and UML. It is certainly worthwhile to investigate a more profound integration of language mapping and method engineering and how these fields could benefit from each other. There is some evidence that language mapping can be used in other method engineering strategies, too, and the mapping process might be supported by method engineering techniques such as the association technique.

Finally, our chapter only scratches at the surface of a fundamental research question that deserves more attention: How can we align the processes of organizational development and systems development in such a way that the information systems support the business in an optimal way? This question is studied under the heading "co-design (or co-evolvement) of business and IT," but so far we have only seen the very beginnings of research in that area.

CONCLUSION

The ideas in this chapter were inspired by a project we carried out 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, and so forth. 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 for 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 suggests an approach to solving the first problem. It allows for the mediation between two important 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 deeply rooted in linguistics and the study of human communication, while UML has many of its roots in computer science and the study of software artefacts (though by far not all). It is therefore surprising that a language mapping can be undertaken at all. It should be noted, though, that we have chosen from the set of all language-action approaches the one that best facilitates integration with UML. Other languages of the action view might prove to be less suitable. Nevertheless, we hope that our work can contribute to narrowing the gap between organizational modeling and the design of information systems.

REFERENCES

Austin, J. L. (1962). *How to do things with words*. Oxford University Press.

Caplat, G., & Sourrouille, J. L. (2003, October 21). Considerations about model mapping. In J. Bezivin & M. Gogolla (Eds.), *Workshop in software model engineering (WiSME@UML '2003)* San Francisco. Online version available at http://www.metamodel.com/wisme-2003/18.pdf

Denning, P. J., & Medina-Mora, R. (1995). Completing the loops. *Interfaces, 25*(3), 42-57.

Dietz, J. L. G. (1999). Understanding and modeling business processes with DEMO. In J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, & E. Métais (Eds.), *Proceedings of Conceptual modeling—ER '99 (Lecture notes in computer science 1728)* (pp. 188-202). Berlin: Springer.

Dignum, F., & Weigand, H. (1995, June 12-16). Modelling communication between cooperative systems. In J. Iivari, K. Lyytinen, & M. Rossi (Eds.), *Advanced information systems engineering, Proceedings of the 7th International Conference, CAiSE '95*, Jyväskylä, Finland (Vol. 932, pp. 140-153). Lecture Notes in Computer Science Berlin: Springer.

Dobing, B., & Parsons, J. (2006). How UML is used. *Communications of the ACM, 49*(5), 109-113.

Dori, D. (2002). *Object-process methodology. A holistic systems paradigm*. Berlin: Springer.

Evans, A., Maskeri, G., Sammut, P., & Willans, J. S. (2003, October 21). Building families of languages for model-driven system development. In J. Bezivin & M. Gogolla (Eds.), *Workshop in Software Model Engineering (WiSME@UML'2003)*. San Francisco Online version available at http://www.metamodel.com/wisme-2003/ 06.pdf

Goldkuhl, G. (1996). Generic business frameworks and action modelling. In F. Dignum, J. Dietz, E. Verharen, & H. Weigand (Eds.), *Communication Modeling—The Language/Action Perspective, Proceedings of the First International Workshop on Communication Modeling, Electronic Workshops in Computing*. Berlin: Springer.

Goldkuhl, G., & Röstlinger, A. (1993). Joint elicitation of problems: An important aspect of change analysis. In D. Avison, J. Kendall, & J. DeGross (Eds.), *Human, organizational, and social dimensions of information systems development*. Amsterdam: North-Holland.

Habermas, J. (1984). *The theory of communicative action 1, Reason and the rationalization of society*. Boston: Beacon Press.

Johannesson, P. (1995). Representation and communication: A speech act based approach to information systems design. *Information Systems, 20*(4), 291-303.

Lehtinen, E., & Lyytinen, K. (1986). An action based model of information systems. *Information Systems, 11*(4), 299-317.

Lodder, A. R., & 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). New York: ACM.

Medina-Mora, R., Winograd, T., Flores, R., & Flores, F. (1992). The action workflow approach to workflow management technology. In J. Turner & R. Kraut (Eds.), *Proceedings of the Conference on Computer-Supported Cooperative Work, CSCW'92*. New York: ACM Press.

Miller, J., & Mukerji, J. (Eds.). (2003). *MDA guide* (version 1.0.1). Needham: OMG. Online version available at http://www.omg.org/docs/omg/03-06-01.pdf

OMG (2004). *Enterprise collaboration architecture specification* (version 1.0). Needham: OMG. Online version available at http://www.uml.org

Ralyté J., Deneckère, R., & Rolland, C. (2003, June 16-18). Towards a generic model for situational method engineering. In *Proceedings of the15th International Conference on Advanced Information Systems Engineering (Caise 2003)*, Klagenfurt, Austria, (pp. 95-110). Heidelberg, Germany: Springer-Verlag.

Reijswoud, V. E. van (1996). *The structure of business communication: Theory, model and application*. PhD Thesis. Delft, The Netherlands: Delft University of Technology.

Reijswoud, V.E. van, & Dietz, J. L. G. (1999). *DEMO modelling handbook* (volume 1). TU Delft. Online version available at http://www.demo.nl/documents/handbook.pdf

Searle, J. R. (1969). *Speech acts, an essay in the philosophy of language*. London: Cambridge University Press.

Weigand, H., & van den Heuvel, W. J. (1998). Metapatterns for electronic commerce transactions based on FLBC. In *Proceedings of the Hawaii Int. Conf on System Sciences (HICSS '98)* IEEE Press.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design*. Norwood, NJ: Ablex.

KEY TERMS

Action View: In the action view, a system consists of a number of actors that can act of their own accord motivated by internal needs and desires as well as react to stimuli from the environment or other actors.

Actor: An actor is a human being. An actor can act on his/her own behalf or on the behalf of some organization.

Agent: An agent is an artefact, that is, inanimate, but can nevertheless exhibit behaviour. This behaviour is pre-determined and follows a given set of instructions which involve suitable reactions to external stimuli.

Concept Mapping: In concept mapping, we identify concepts in the target language that match the concepts of the source language. This is not always a one-to-one correspondence and may involve complex and structured relations.

Diagram Transformation: Models are typically represented as diagrams, that is, graphs. Graph conversion rules are employed to translate the source diagram into the target diagram.

Language Mapping: In language mapping, we "translate" a model from one language to another while preserving as much as possible of the original semantics. The steps involved in this process are concept mapping, notational mapping, and diagram transformation.

Notational Mapping: Notational mapping resolves conflicts arising from different notations (symbols) used for the same or similar concepts in source and target language.

Reaction View: In the reaction view, the system consists of a number of agents that merely react to stimuli from the environment or other agents.