Business Conversations
A High-Level System Model for Agent Coordination

Florian Matthes

AB 4-022 Softwaresysteme
Technical University Hamburg-Harburg
D-21071 Hamburg, Germany
f.matthes@tu-harburg.de

Abstract

In this paper we introduce Business Conversations as a high-level software structuring concept for distributed systems where multiple autonomous agents (possibly in different organizational units) have to coordinate their long-term activities towards the fulfillment of a cooperative task. We first motivate Business Conversations as a system model suitable for the description of human-human, human-software as well as software-software cooperation. We then explain why we consider this model be more suitable for the description o organizational cooperative work than software-centered object models. The core concepts of the Business Conversation model are described using an object-oriented model. Finally, we report on our experience gained building a prototypical agent programming framework with Business Conversations for agent coordination based on mobile and persistent threads as provided by the persistent programming language Tycoon.

1 Motivation and Background

Following [De Michelis et al. 97] a cooperative information system (COOPIS) can be described using three complementary facets: A system integration facet which addresses data transfer as well as semantic and control integration, a group collaboration facet which is concerned with how people working on a common business process can coordinate their activities, and an organizational facet which views cooperation from a formal organizational perspective, regardless by whom or with what technology it is carried out.

In this paper we propose a high-level system model for the system integration facet where distributed autonomous (human or software) agents coordinate their long-term activities towards the fulfillment of a cooperative task via structured Business Conversations. In our prototypical implementation of this model agents and conversations are treated as first-class entities: They can be named, classified, specialized, made persistent and they can migrate across heterogeneous execution and communication platforms.

As should become clear in the remainder of this paper, this system integration model fits well with modern models for the group collaboration facet (business process modeling, workflow management, groupware support systems) and the organizational facet (virtual enterprises, electronic commerce, radically decentralized information processing). Our model therefore has the potential to simplify the central task of *change propagation* between the facets of a larger-scale COOPIS.

To our understanding, any model developed for the system facet should allow system builders to abstract from the details of

1. cooperation over **time**: Agents and artifacts should exist as long as required by the business processes they support, independently of the underlying language and system concepts, e.g. the lifetime of operating system processes or of database schema revisions.

2. cooperation within **space**: Agents and artifacts should be able to migrate freely within a physically distributed environment, independently of the particular system platforms or organizational structures (e.g. business units) involved in the cooperative work.

3. cooperation in multiple **modalities**: Artifacts should be accessible uniformly for agents that cooperate in different modalities (simple overnight batch processing, online transaction processing, direct manipulation by human agents via form-based or graphical user interfaces, computer-supported cooperative work by humans, computer-assisted workflow management, automatic information processing by mobile software agents) and using different media (email messages, EDI-messages, HTTP requests, CORBA object requests, RPC invocations).

As a consequence, a system builder can focus on the "what" and "how" of the particular problem at hand (e.g. compilation of an insurance offer) without worrying about the "when" (batch processing, interactive session, long-term conversation with a human customer), the "where" (on a centralized host, on a server within sales department, on a mobile laptop of a salesperson) and the "who" (customer via letter, salesperson, workflow management system, visit of an Internet agent). Moreover, applications are not cluttered with unnecessary data movement, synchronization and conversion details and can absorb much better changes in time, platform and modality than it is the case today.

Our research work of the past three years has focused on the issues (1) and (2) by providing full **persistence** and **mobility** abstraction in the platform-independent Tycoon system [Matthes, Schmidt 94; Mathiske et al. 95] culminating in the development of persistent migrating threads [Mathiske et al. 96] which can be used to implement software agents similar to Telescript agents [Mathiske 96; Johannisson 97]. In this paper, we describe Business Conversations as a contribution to the third issue, cooperation in multiple modalities. Even though Business Conversations are implemented in the Tycoon environment, this high-level system model has a much wider applicability since it makes little assumptions about the underlying agent infrastructure and (persistent) programming model.

This paper is organized as follows: In Section 2 we position our activity-oriented approach to system cooperation relative to other, more traditional, data-oriented and object-oriented models. In Section 3 we give a high-level overview of our Business Conversation model using examples from the insurance sector. Details of this model are then given in Section 3 which also discusses our prototypical implementation of the model. The paper ends with a comparison with related work and an outlook on future work.

2 System Models for Cooperative Business Applications

In this section we motivate the need for a high-level, activity-oriented agent coordination model and also relate our work to other models for cooperative business applications.

At a certain level of abstraction, the evolution of system models for cooperative business applications can be classified roughly into three stages. Within each of these "historic" stages, the primary focus of research and development has been on a certain aspect of business applications, namely data, behavior and activity. As a consequence, one can distinguish roughly between data-oriented, object-oriented and agent-oriented approaches for the construction of distributed business applications.

2.1 Distributed Business Data

The sharing of business data between multiple applications of an enterprises has been the first approach to the construction of large cooperative information systems (banking systems, airline reservation systems, management information systems, CIM systems). A cooperation of multiple applications is supported by persistence and distribution abstraction as provided, for example, by modern relational database systems. Due to the failure of distributed database technology, most

existing cooperative IS are based on centralized data servers, as exemplified by integrated business application systems like SAP R/3, Baan or Oracle Financials.

In a nutshell, such cooperative information systems are viewed as collections of shared database servers and distributed database clients (application programs or ad-hoc human users).

2.2 Distributed Business Objects

The promise of distributed object management is to arrive at more flexible, scaleable and maintainable system architectures by building cooperative information systems using distributed business objects [Orfali et al. 96]. A business object encapsulates business data and achieves a higher degree of autonomy by restricting access to the business data through well-defined method interfaces. Clients of a business object not only can abstract from the internal representation of the business information but they can also rely on an open communication platform (middleware) that achieves full distribution and platform transparency, possibly also across organizational boundaries. Moreover, it is expected that there exists a refinement relationship between business objects (e.g., a car insurance contract is a refinement of a general insurance contract which is in turn a refinement of a generic customer contract) which can be exploited to develop generic domain-specific application models or to define standardizable business APIs.

In the Distributed Business Object model, a cooperative information system is viewed as a collection of distributed objects which exchange messages that are dispatched by an "ubiquitous" object request broker infrastructure.

In our view, Distributed Business Objects are a promising software structuring concept for rather tightly integrated business applications, e.g. in-house desktop clients accessing corporate business object servers. However, we believe that it does not scale well for more advanced patterns of cooperative work involving truly autonomous "profit centers" within an organization or involving several departments of independent enterprises which are unlikely to agree on a common business object model and a shared object infrastructure which may be expensive to maintain.

The interaction between business objects and the coordination of their behavior is "hard-coded" and often distributed in a complex way over the methods of multiple objects. This is to be seen in contrast with the need for multiple modalities for cooperative work quoted in the introduction of this paper and the flexibility requirements imposed by business process reengineering [De Michelis et al. 97].

2.3 Distributed Business Agents

In recent years, activity-oriented models centered around the notion of "agents" have attracted a lot of interest in the research community. Despite significant differences in detail, all of these models suggest to view cooperative systems as being composed of largely autonomous agents, each with a well-defined "responsibility", "independent activity", private "knowledge", "memory" and "capabilities". Moreover, agents are to be regarded as a unit of persistence and mobility.

Agents provide a system structuring concept appropriate for the decomposition of cooperative systems into smaller subsystems responsible for well-defined short-term or long-term tasks (claims processing, order management, shopping, information retrieval) which can be carried out either by a human or a software agent (demon, robot, script invocation, etc.) at a single site or involving a migration from site to site.

In order to avoid the deficiencies of business objects described at the end of the previous section, the notion of a business agent should not simply expand the notion of a business object (consisting of encapsulated state and behavior) by adding autonomous activity (active objects) and platform-independent mobility as suggested by some software agent models.

Instead of this, our proposal de-emphasizes the importance of agents themselves (how they are created, classified, duplicated, etc.) and concentrates on their externally observable behavior, namely their ability to sustain long-term, goal-directed conversations with other agents which is also an important mechanism to *coordinate* agents (synchronization, delegation, replication, ...).

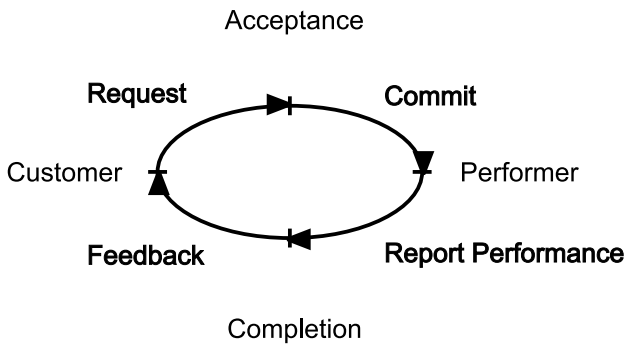

Figure 1: The four phases of a Business Converation

Another view on the evolution of cooperation models is to regard it as a gradual generalization process, since each of the newer models subsumes concepts of its predecessors and puts them into a larger context which tends to be more stable over the lifetime of a cooperative information systems.

- Data-centered modeling: Which data structures are maintained by the system? What are their attributes and relationships? Which integrity constraints exist on these data structures?
- Object-centered modeling: Which operations can be applied to these data structures? What are legal state transitions on these objects?
- Activity-centered modeling: How does the system interact with its environment over time? Are there subsystems with restricted communication links to other subsystems? What is the long-term goal to be achieved by a sequence of communication steps between subsystems?

3 Concepts of the Business Conversation Model

The leitmotiv of the Business Conversation model are speech acts between customers and performers. The model is inspired by the work of Terry Winograd and Fernando Flores in the domain of computer-supported cooperative work [Winograd 87; Flores et al. 88; Medina-Mora et al. 92] which introduce the concept of "conversations for action" based on linguistic studies and the speech act theory developed by Austin and Searle [Austin 62; Searle 69]. However, our goals are different from the more descriptive work of Winograd and Flores in that we intend to develop a *system model* and system design techniques which can be used during system analysis, system design and system implementation (similar to the work on distributed object models).

3.1 Business Conversations of an Enterprise

In the Business Conversation model, an enterprise or a business unit is viewed as an agent that is involved in a number of (long-term) business conversations with other agents like customers, suppliers or government agencies. Within each of these conversations, each agent has a fixed role (either customer or performer). For example, an insurance broker is a performer for its customers and at the same time a customer for several insurance agencies.

The main purpose of a business conversation is to coordinate the otherwise autonomous activities of both agents towards a common goal which is typically specified in the course of the conversation. A business conversation can be decomposed into an ordered sequence of speech acts which can be classified into four phases that occur in the following order (see also Figure 1):

- Request Phase: During this conversation phase, the customer identifies himself to the performer and states its (business) goal to be achieved during the conversation. The main

purpose with respect to the customer is to check whether the performer is ready to start such a conversation or not. ("I want to insure my car").

- Negotiation Phase: A sequence of negotiation speech acts may be necessary to align the specific customer needs and the available performer services. Only if both partners (as autonomous agents) agree on the (refined) common goal, a commit of the performer is reached which can be understood as a "promise" about its future activity. During the negotiation phase, exception handling and recovery policies (or more general "quality of service" parameters) for the remainder of the conversation can be specified. ("Here are our insurance policies", "Your insurance contract number is 12345").
- Performance Phase: The performer reports on the progress of and/or the completion of the requested activity to the customer. This may also involve requests for additional information or actions from the customer to take place under the already committed QoS conditions. ("'We covered your last accident", ...)
- Feedback Phase: This phase gives the customer the opportunity to declare its satisfaction with the service provided and may comprise the obligation for payment. During the feedback phase, no further services have to be provided by the performer.

If required by a specific real-world cooperative activity, some of these phases may be skipped. For example, a simple atomic client/server transaction (money bank transfer) involves neither a negotiation nor a feedback phase.

3.2 Cooperation in Multiple Modalities

The utterances of customers and performers quoted in the preceding section are deliberately chosen to resemble natural language statements of human actors. However the Business Conversation model and the system infrastructure described in Section 3 are intended to cover uniformly human as well as software agents. Given a formal conversation specification for a specific business task such as claims handling, it can be utilized directly to support agent cooperation in four modalities

Application Linking: Customer and performer are realized as two autonomous applications that synchronize via asynchronous message exchange.

GUI Management: A human user interacts with a software system. Legal interaction patterns are described by the Business Conversation specification which are interpreted by a software system called "generic customer".

Workflow Management: A software system (as a customer) requests actions from a human user who is guided by a software system called "generic performer".

Structured Message Handling: The cooperative work of two human users can also profit from tool-supported message handling ensuring the adherence to pre-defined business rules.

An interesting technical implication of this unification of human and software agents is the ability to first develop a stand-alone information system based on its externally observable interaction patterns (similar to Visual Basic or Visual Age) and then to scale this application to a richer distributed environment where it is necessary to cooperate with other information systems without modifying the application logic of the individual systems.

Moreover, we are currently building generic application wrappers for legacy applications (Windows-95 applications and SAP R/3) which translate automatically conversation specifications and dialog steps to support the integration of such applications (without access to their source code) as agents into an open cooperation environment.

Finally, it should be noted that the modality may change dynamically during an ongoing conversation. For example, similar to an automated telephone call center, standard requests could be handled by a software agent which transfers its conversation (i.e. a simple trace of past communication steps) to a human performer as soon as more complex or exceptional requests occur.

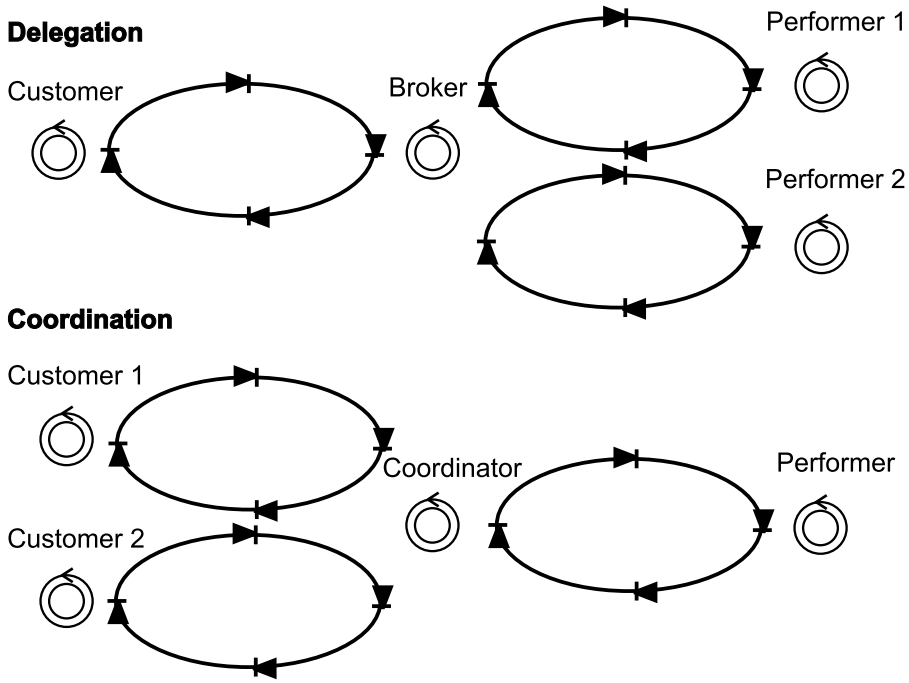

Figure 2: Interactions between conversations: delegation and coordination

3.3 Refinement and Abstraction

An externally observable "customer-oriented" business conversation (primary business conversation) of an enterprise may lead to other internal business conversations (secondary business conversations) between autonomous business units of an enterprise.

A secondary business conversation follows the same communication pattern described for primary conversations. For example, a performer in a primary conversation may become a customer in multiple secondary conversations to coordinate the cooperative work of multiple subordinate agents towards the common goal requested by the external customer in the primary conversation.

Such a situation is displayed in the upper part of Figure 2, where the external customer establishes a conversation with a performer which delegates the customers requests transparently to the respective subordinate performer specialized on this particular kind of task (claims processing, assessment of damage) and reports their utterances back to the customer of the primary conversation.

An agent can also take the performer role in multiple conversations simultaneously (see the lower part of Figure 2), hiding coordination details (planning, prioritizing, ...) in cases where the customers request activities on shared resources.

Similarly, an "external" customer may consist of a collection of cooperating subordinate agents (secretary, deputy) which cooperate through (externally invisible) means like peer to peer business conversations or simple authority chains (from boss to employee).[1]

To summarize, the iterated decomposition of a binary customer/performer conversation into secondary conversations or subordinate agents ultimately terminates with a collection of human or software agents which carry out their work tasks in total autonomy but which are linked by a tree of speech acts (requests, commitments and task completion dependencies) and by an authority hierarchy.

[1] In the remainder of this paper we ignore the implications of the concept of subordinate agents which participate in business conversations on behalf of their authority (security, addressing, mobility, ...).

3.4 Structured Dialogs and Conversation Specifications

In our model, the speech acts (protocols) between customer and performer are constrained to be *structured dialogs* and they have to adhere to explicit *conversation specifications*.

Two adjacent speech acts of a business conversation are grouped together and form a dialog step which involves the exchange of a dialog understood as a document with a hierarchically structured content according to the following basic pattern: The performer sends a dialog with an initial content, for example, a list of insurance offers with a partially filled-out contract that contains placeholders to prompt the customer for additional information. (3.) The customer updates its copy of the dialog (i.e., "fills out the form") and returns it to the performer with a request from a set of possible requests available in this particular dialog step (e.g. "Please revise the offer" or "I accept the offer").

A conversation is initiated by the customer with an initial request consisting of a description of the conversation specification (protocol) the customer will utilize for the remainder of the conversation. This description is represented as a structured dialog itself.

The conversation terminates successfully only when both agents agree that the conversation is completed. In each dialog step, a time-out can be regarded as an implicit request (of the customer) or as an implicit reply (of the performer), respectively.

Each structured document exchanged between customer and performer is required to contain a document header that permits the receiving agent to uniquely identify the context of the dialog (i.e. the enclosing conversation). As a consequence, the very first request in a conversation uttered by the customer ("I want to start a new conversation conforming to specification X") has to be different from all its subsequent requests which are shipped together with the revised content of its previous dialog step and a reference to an already existing conversation.

The advantage of imposing additional structure onto conversations and of constraining conversations by structured meta data (conversation specifications) can be best understood by the analogy with the concept of data and object types which also provide a stable basis for formal reasoning, for tool support (consistency checking) and for system analysis (classification, generalization, parameterization).

Moreover, this meta information (protocol information) is readily available (business rules, workflow specifications, object interaction diagrams) at system design time and is essential for the understanding of large reactive agent systems. [2]

4 Implementing Business Conversations

This section gives an overview of our prototypical implementation of the business conversation model sketched in the previous section. This implementation consists of a polymorphically-typed framework written in the Tycoon persistent and distributed programming environment [Matthes et al. 97], exploiting mobile persistent threads described in [Mathiske et al. 96]. The framework components are replicated at each agent-enabled network site and are therefore viewed as an "ubiquitous" infrastructure.

The three layers of this library framework are depicted in Figure 3:

Application system layer: This layer contains the application-specific agent definitions ("domain-specific business logic") consisting of Business Conversation specifications with attached event specifications which in turn are bound to statically scoped and typed Tycoon application code.

Agent system layer: The API provided by the agent system layer is used by the agents to interact with their environment. This layer is responsible for the coordination of concurrent

[2] "An object-oriented program's run-time structure often bears little resemblance to its code structure. The code is frozen at compile-time, it consists of classes of fixed inheritance relationships. A program's run-time structure consists of rapidly changing networks of communicating objects. Trying to understand one from the other is like trying to understand the dynamism of living ecosystems from the static taxonomy of plants and animals, and vice versa". [Gamma et al. 95]

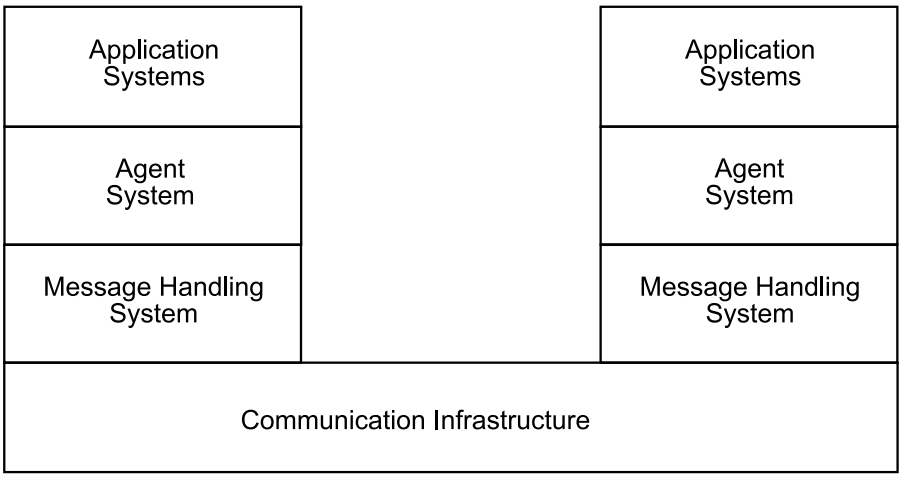

Figure 3: Layers of the Business Conversations library framework

conversations and the tracing of conversation instances. Furthermore, agent mobility and persistence services are localized in this layer.

Message handling layer: This layer provides a programming abstraction from local and remote communication implementing a store-and-forward messaging scheme. Message queues ensure the correct transmission of a message even in cases where the receiver is temporarily unavailable [MSMQ95 95].

All interactions between agents at higher layers are carried out via this component. Furthermore, the message handling system provides an abstraction of the underlying communication infrastructure. As of today, simple (secure) Internet socket connections are used to transfer linearized typed Tycoon objects. We are currently experimenting with software components for automatic forms processing (FAX communication channel) and with SGML (in particular HTML) converters as commercially-relevant alternatives for communication in open networks with legacy applications.

In the following sections we focus on the object types and services provided by the agent system layer to the application systems.

4.1 Conversation Specifications as First-Class Objects

A conversation specification is a contract between two agents since it constrains the behavior of the performer and provides a promise to the customer. A conversation specification can express type constraints (the structure of the documents sent and received) but also as state-dependent constraints on the conversation history (e.g., claims can only be settled after a contract has been signed and the first payment has been received). A more software-oriented example is the state-dependent specification that a customer will never execute a *pop* operation on an empty stack.

In contrast to earlier models based on the speech act model like the Coordinator Tool [Flores et al. 88], a conversation specification is not given as part of the business conversations model. Instead of this, a conversation specification is a dynamically-created structured object which describes the set of all possible conversations between a customer and a performer in a given application domain expressed in the syntax of the business conversation model.

The object model in Figure 4 defines the abstract elements of conversation specifications using the Mainstream Objects Model as a meta model notation [**?**] (IH = inheritance, A = aggregation. Circle = cardinality zero, double line = optional, crow foot = cardinality n).

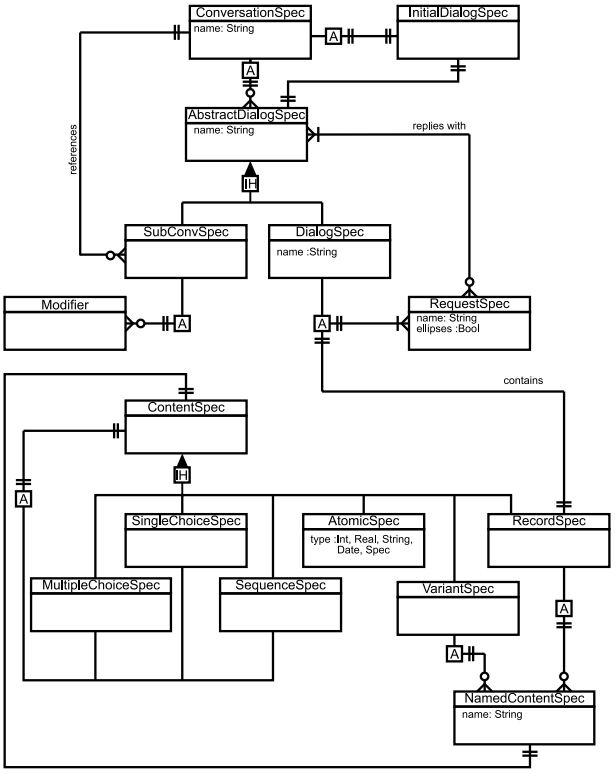

Figure 4: Class diagram for content, dialog and conversation specifications

Technically speaking, conversation specifications are typed, persistent and mobile objects that are created bottom up using constructors of their respective classes (compare Figure 4), for example:

let *contract = RecordContentSpec.new()*
.add("name" AtomicContentSpec.new(String))
.add("first" AtomicContentSpec.new(String))
.add("birthday" AtomicContentSpec.new(Date))
.add("method of payment" MultipleChoiceSpec.new().add(...))
let *contractDialog = DialogSpec.new(contract)*
.addPossibleRequest("Accept" #(confirmationDialog))
.addPossibleRequest("Reject" #(negotiationSubConversation, noAgreementDialog))
let *carInsuranceConversationSpec = ConversationSpec.new("carInsurance")*
.add("Welcome" welcomeDialog)
.add("Contract" contractDialog)
...

It is also possible to generate a conversation specification from a textual or graphical representation of the dialog graph or to receive it from a remote agent, for example, a "conversation broker".

A conversation specification ("car insurance") is a dictionary of named dialog specifications with a distinguished initial dialog specification which describes the initial state for both communication partners. A dialog specification can be are either a concrete dialog specification ("contract") or a subconversation specification ("negotiation on contract details"). The latter is obtained by an incremental modification of an existing conversation specification. This incremental modification concept is intended to generalize the concept of inheritance on object classes to conversation specifications.

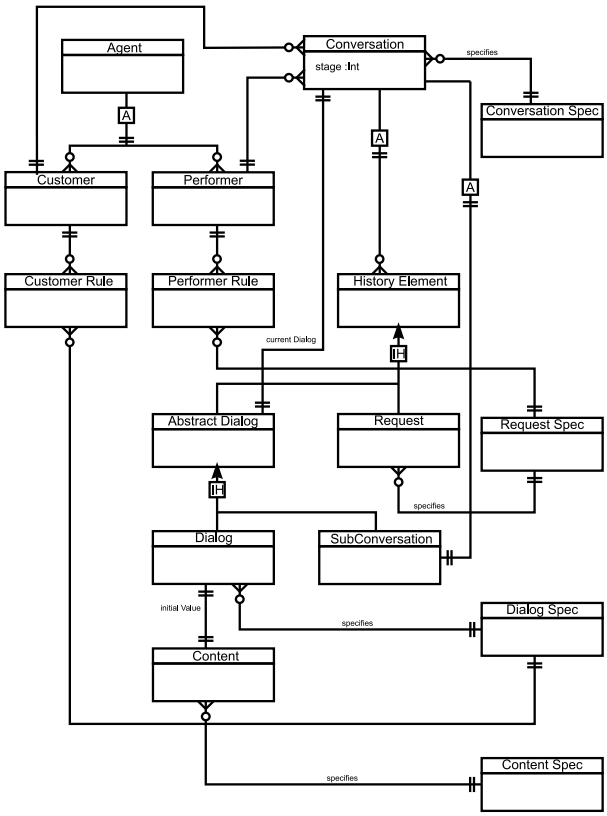

Figure 5: Class diagram for agents, roles and conversation instances

A concrete dialog specification consists of a record dialog content specification and a (possibly empty) set of request specifications available for this dialog. A record dialog content specification aggregates named content specifications ("name", "birthday", "method of payment") which in turn (and recursively) can be either atomic (integer, string, ...), record, variant, sequence, single choice and multiple choice content specifications. In this way, content specifications define a simple monomorphic type system. More details on the Tycoon implementation can be found in [Johannisson 97].

A dialog specification can be understood as a labeled state of a non-deterministic finite automaton. The states of a conversation are connected by directed edges, each labeled by a request (uttered by the customer, e.g. "reject contract"). In a given dialog, there may be multiple outgoing edges with the same label. This nondeterminism makes it possible for the performer to choose between multiple follow states ("no agreement possible", "start with negotiation").

4.2 Event-Based Software Agent Programming

Once a Business Conversation specification object has been created, performer and customer agents which adhere to this specification can be defined by rules consisting of an event and a piece of code. This code typically triggers state transitions, initiates secondary conversations or performs actions through effectors attached to the agent. The code is parameterized by a conversation descriptor which holds conversation-specific data (identity of the conversation partner, contents and requests of all preceding dialog steps, etc.). This descriptor can be expanded by agent-specific data (invisible to other agents) and greatly simplifies the management of concurrent conversations with multiple customers and performers, respectively. On termination, the code attached to an event has to return an object that matches the constraints expressed in the corresponding conversation specification which is then transferred back to the customer.

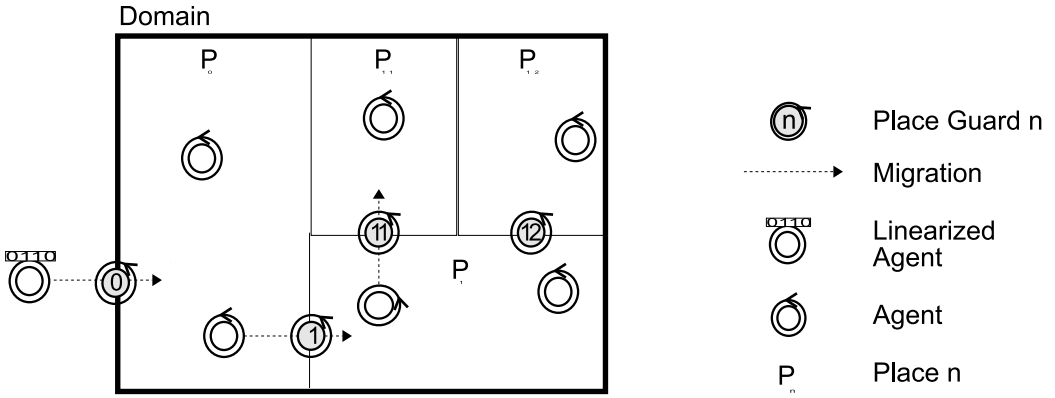

Figure 6: Logical structure of the agent world

An agent can support multiple customer and performer roles (e.g. performer for car insurance, performer for freight insurance). For each of these roles there exists a set of customer and performer rules, respectively.

A performer rule is defined for a particular request of a particular dialog specification (contract.accept, to be issued by a an agent in a customer role) and has to return an object of class dialog while a customer role is defined for a particular dialog specification (e.g., contract, to be generated by an agent in the performer role) and has to return an object of class request which has to be one of the requests admissible in this dialog step (e.g., accept, reject, explain).

The matching between dialog and dialog specification is checked by the agent system which gives a special treatment to the distinguished initial request of a conversation ("I want to start a new conversation of type X") and a predefined "breakdown" request which is allowed to occur in any state of the conversation.

An active conversation links exactly one customer role and one performer role of an agent. It also implements exactly one conversation specification and aggregates an ordered list of history elements which record the past dialog steps and requests of this conversation.

Figure 5 summarizes the semantic relationships between the relevant classes, some of which have already been introduced in Figure 4.

Customer and performer neither share data nor code or thread state. In particular, the conversation specification and the conversation trace are duplicated in the address spaces of both communication partners which ensures a high degree of agent autonomy and mobility.

The generic customer and performer described in Section 3.3 are also implemented using this event-based execution model and simply transfer incoming and outgoing information to a human agent via a (HTML-based) user interface.

4.3 Agents, Places and Place Guards

Our prototypical execution environment for Business Conversation agents incorporates a rich spatial world metaphor for the "agent world" which is similar to the one of Telescript based on the concepts of *domain* and *place* (see Figure 6). A domain is associated with a node in a network and has its own communication end point on the network. A domain is uniquely addressable in the network and consists of at least one place which constitutes the root of the local place hierarchy of this domain. A place can host an arbitrary number of other places and agents for which it provides a dynamic name space. Each place is represented by a specialized agent, a so-called place guard. This guard can selectively grant access to agents which request to enter its place through agent migration.

An agent can establish conversations with agents at its own place. A place (resp. its place guard) can be accessed from within its place as well as from other places and it is therefore positioned on the borderline of its place in Figure 6. Agent migration is controlled by a path consisting of the

network domain of the destination domain and of the names of all places on the way to the target place. Local migration can be controlled by relative path names.

By treating agents and places (via their place guards) uniformly, both utilize the business conversation model for cooperative work. In particular, a migration conversation consists of a migration request of an agent, followed by a negotiation with its place guard, a subconversation with the target place guard(s) carried out by the place guard, a completion report and finally a declaration of satisfaction of the agent as soon as it resumes execution at the receiver side.

Similarly, the core model makes very little assumptions about the mechanisms through which a customer localizes a matching performer in a wide-area network. This well-known problem of *trading* or *brokering* in client/server programming can be solved above the core agent system layer by transmitting conversation specifications as part of ordinary trading conversations.

4.4 Comparison with Related Work

Dialog-oriented cooperation based on the Business Conversation model has several advantages over cooperation based on direct object interaction (message passing) which underlies popular agent system like Telescript [General Magic 95], Mole [Hohl 95], Facile [Knabe 95], COSY [Haddadi 95], Tcl/MIME agents [Rose 93] and Obliq [Cardelli 94]:

- Conversations do not impair agent autonomy. By exchanging well-defined dialog content with copy semantics only, no private object bindings become available to the communication partner. Therefore, it is not necessary to introduce new binding mechanisms like the ill-defined concept of "object references" described in [White 94; GMI95 95] which attempts to distinguish objects belonging to the customer and the performer, respectively.

- Conversations do not restrict agent mobility since local and remote agents are treated uniformly. Therefore, it is possible for an agent to migrate between address spaces while sustaining long-term conversations, e.g. with its human "owner".

- The agent system already provides a well-defined concurrent execution model. Contrary to direct object interactions, the coordination of multiple agents and conversations is encapsulated in the agent system layer and there is no necessity for application-level synchronization in cases where shared resources are manipulated. The application programmer is thus shielded from much of the complexity that arises in highly concurrent agent systems

- The details of the information exchange between agents are to a large degree independent of the underlying communication infrastructure. In particular, it is possible to translate existing object-based distribution mechanisms (e.g., CORBA [Otte et al. 96]) systematically into Business Conversations.

- Conversations are based on static process descriptions (conversation specifications) which are first-class run-time objects available to both communication partners as soon as a conversation is initiated. This makes it possible to detect mismatches between the customer and performer view early.

4.5 Concluding Remarks

We have presented a high-level coordination model for autonomous agents which is tailored to the needs of (cross-enterprise) business information systems and which is excelled by its uniform treatment of human and software agents. The model exhibits several advantages over other agent models and can be implemented with limited effort in modern platform-independent, distributed and persistent programming languages like Tycoon, Napier and P-Java.

Open issues include a formalization of the refinement relationship between conversation specifications (subtyping) and of their incremental modification by overriding (inheritance). Moreover, we plan to investigate whether it is possible to build a static type checker that guarantees that a

software agent generates (at run-time) only conversation traces which conform to a given static conversation specification.

References

Austin 62: Austin, J. *How to do things with words.* Technical report, Oxford University Press, Oxford, 1962.

Cardelli 94: Cardelli, L. *Obliq: A Language with Distributed Scope.* Technical report, Digital Equipment Corporation, Systems Research Center, Palo Alto, California, Juni 1994.

De Michelis et al. 97: De Michelis, Giorgio, Dubois, Eric, Jarke, Matthias, Matthes, Florian, Mylopoulos, John, Papazoglou, Mike, Pohl, Klaus, Schmidt, Joachim, Woo, Carson, and Yu, Eric. *Cooperative Information Systems: A Manifesto.* In: Papazoglou, Mike P. and Schlageter, Gunther (Eds.). *Cooperative Information System: Trends and Directions.* Academic Press, 1997.

Flores et al. 88: Flores, F., Graves, M., Hartfield, B., and Winograd, T. *Computer Systems and the Design of Organizational Interaction.* ACM Transactions on Office Information Systems, Jg. 6, 1988, Nr. 2, S. 153–172.

Gamma et al. 95: Gamma, E., Helm, R., Johnson, R., and Vlissades, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company, 1995.

General Magic 95: General Magic's Telescript home page . http://www.genmagic.com/Telescript/, 1995.

GMI95 95: General Magic, Inc. *Telescript Devloper Environment, Version 1.0 alpha*, Oktober 1995. Internet WordWideWeb, see General Magic homepage.

Haddadi 95: Haddadi, Afsaneh. *Communication and Cooperation in Agent Systems.* Springer-Verlag, 1995.

Hohl 95: Hohl, Fritz. *Konzeption eines einfachen Agentensystems und Implementation eines Prototyps.* Diplomarbeit, Universitaet Stuttgart, Abteilung Verteilte Systeme, August 1995.

Johannisson 97: Johannisson, Nico. *An environment for mobile agents: agent-oriented distributed databases.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, Germany, April 1997. (In German).

Knabe 95: Knabe, Frederick Colville. *Language Support for Mobile Agents.* Dissertation, Carnegie Mellon University, Pittsburgh, PA 15213, Oktober 1995.

Mathiske et al. 95: Mathiske, B., Matthes, F., and Schmidt, J.W. *Scaling Database Languages to Higher-Order Distributed Programming.* In: *Proceedings of the Fifth International Workshop on Database Programming Languages, Gubbio, Italy.* Springer-Verlag, September 1995. (Also appeared as TR FIDE/95/137).

Mathiske et al. 96: Mathiske, B., Matthes, F., and Schmidt, J.W. *On Migrating Threads.* Jg. 8, 1996, Nr. 2. Journal of Intelligent Information Systems.

Mathiske 96: Mathiske, B. *Mobility in Persistent Object Systems.* Dissertation, Fachbereich Informatik, Universität Hamburg, Germany, Mai 1996. (in German).

Matthes et al. 97: Matthes, F., Schröder, G., and Schmidt, J.W. *Tycoon: A Scalable and Interoperable Persistent System Environment.* In: Atkinson, M.P. (Ed.). *Fully Integrated Data Environments.* Springer-Verlag (to appear), 1997.

Matthes, Schmidt 94: Matthes, F. and Schmidt, J.W. *Persistent Threads.* In: *Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB*, Santiago, Chile, September 1994, S. 403–414.

Medina-Mora et al. 92: Medina-Mora, R., Winograd, T., Flores, R., and Flores, F. *The Action Workflow Approach to Workflow Management Technology.* In: Turner, J. and Kraut, R. (Eds.). *Proceedings of the Fourth Conference on Computer-Supported Cooperative Work.* ACM Press, 1992, S. 281–288.

MSMQ95 95: Microsoft Message Queue Server (MSMQ). A White Paper from the Business Systems Technologie Series. Technical report, Microsoft Corporation, 1995. http://www.microsoft.com/msmq/overview.htm.

Orfali et al. 96: Orfali, Robert, Harkey, Dan, and Edwards, Jeri. *The Essential Distributed Objects Survival Guide.* John Wiley & Sons, 1996.

Otte et al. 96: Otte, R., Patrick, P., and Roy, M. *Understanding CORBA: the Common Object Request Broker Architecture.* Prentice Hall, Englewood Cliffs, New Jersey, 1996.

Rose 93: Rose, Marshall T. *MIME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail* . Internet WWW, 1993. working draft.

Searle 69: Searle, J. *Speech Acts.* Technical report, Cambridge University Press, Cambridge, 1969.

White 94: White, J.E. *Telescript Technology: The Foundation for the Electronic Marketplace.* White paper, General Magic Inc., Mountain View, California, USA, 1994.

Winograd 87: Winograd, T.A. *A Language/Action Perspective on the Design of Cooperative Work.* Technical Report Report No. STAN-CS-87-1158, Stanford University, Mai 1987.