INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 05 MELBOURNE, AUGUST 15-18, 2005

SUPPORTING THE RAPID PRODUCT DEVELOPMENT REQUIRMENTS BY A VIABLE SOFTWARE ARCHITECTURE

Emil Stoyanov, Stavros Dalakakis, Dieter Roller, Markus Wischy

Keywords: Rapid Product Development, Product Development Systems, Product Life Cycle, Self-managed Systems, Adaptive Software Architecture, Viable Systems Model

1 Introduction

The Rapid Product Development (RPD) as a technique for design of innovative products is characterized by interaction of development teams distributed in time and space, with different responsibilities, background in various domains, by fast reaction to market changes and by coordination of necessary actions for every development phase between these groups. Nowadays product development is influenced from information technology and informatics with their respective fields of distributed systems, knowledge representation and agent technology. Major challenge in product development systems is the selection of an adequate representation form of knowledge presentation together with the mechanisms for its communication and control. We designed a system architecture, characterized by its integrative and holistic approach, for supporting the domain of RPD. As a promising way for solving problems related to separation of services, finding an intelligent solution and synchronization of design teams we apply a multi-agent system (MAS) as a middleware to an Active Semantic Network (ASN) as a knowledge representation model.

1.1 Motivation

RPD meets the growing problem of increasing complexity in handling product development support systems, knowledge integration and coordination of parallel processes. As a result from the extended impact of the global networking it is compulsory for a system to have a distributed character. This is usually achieved by development of approaches based on loosely-coupled components, transactional mechanisms supporting remote team cooperation and ad-hoc functionality. With increase of a system's scope of application, its shared resources and access interfaces, handling of quality assurance assets such as *stability* and *optimization* takes a considerable time, efforts and human resource. Stability in the process of adaptation and during the whole life-cycle of the supporting runtime is a must. Referencing this, we design system architecture with intrinsic stability features for the needs of RPD. We discuss the topics of knowledge representation in section 2.1, knowledge communication in section 2.2 and focus in details on the adaptation techniques in section 2.3 using cybernetic models and elements from autonomic design for increased stability. While the paper focuses on the proposed system architecture, the concrete used development techniques and standards are pointed as well in section 3.

2 Methods for support of RPD

For the purposes of RPD was designed a software architecture in the frame of research project Sfb374, sponsored by the Deutsche Forschungsgemeinschaft (Figure 1). There are three

abstraction levels with clearly separated purposes. On the lower level stays the ASN Framework which provides a knowledge presentation and storage model. On the second level, serving as a middleware for communication and interaction to the knowledge, stays the Agent Communication Layer. It takes care of the way the RPD user can find a service or another agent and serves his queries by forwarding them to the knowledge base. On the highest level is the RPD User Client which has to handle model realization and present to the user in a friendly way the system's capabilities and methods for interaction with it.

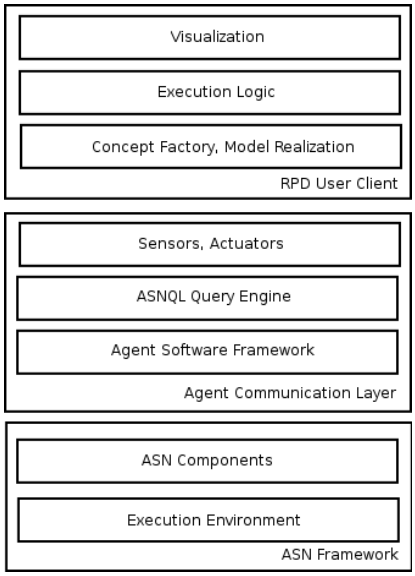

Figure 1. Overview of the software architecture

Within the next paragraphs we present the three layers and we focus on the application of models for enabling adaptation and optimization in RPD process.

2.1 Knowledge Representation

A main idea in knowledge-based systems is the access to knowledge as an external source. The used by the system knowledge is not anymore implicitly stored within a program code but is formed as explicit knowledge base. This split representation enables the users to work with knowledge using different tools. It can be extended, modified or exchanged dynamically, without interaction with the program code. In [1] is described an approach for design and implementation of ASN suited for the purposes of RPD. The architecture includes three levels of abstraction and is shown on Figure 2. The ASN meta-model level provides functionality to model the higher levels and consists of three basic components: Concept, Relation and Attribute. Concepts are the common expression unit by which all entities in the RPD process are identified. Its Attributes that it aggregates describe the way in which properties and features of the RPD objects are expressed. The meta-model level enables the creation of models for various Domains. Here is included on one hand the mechanism for the formation of RPD models and on the other side the metadata structure of the entire system. On the ASN-Model level it comes into representation the knowledge of RPD domain. On the higher run-time level is the presentation of the model, accessible by software programs a as set of dependent object instances.

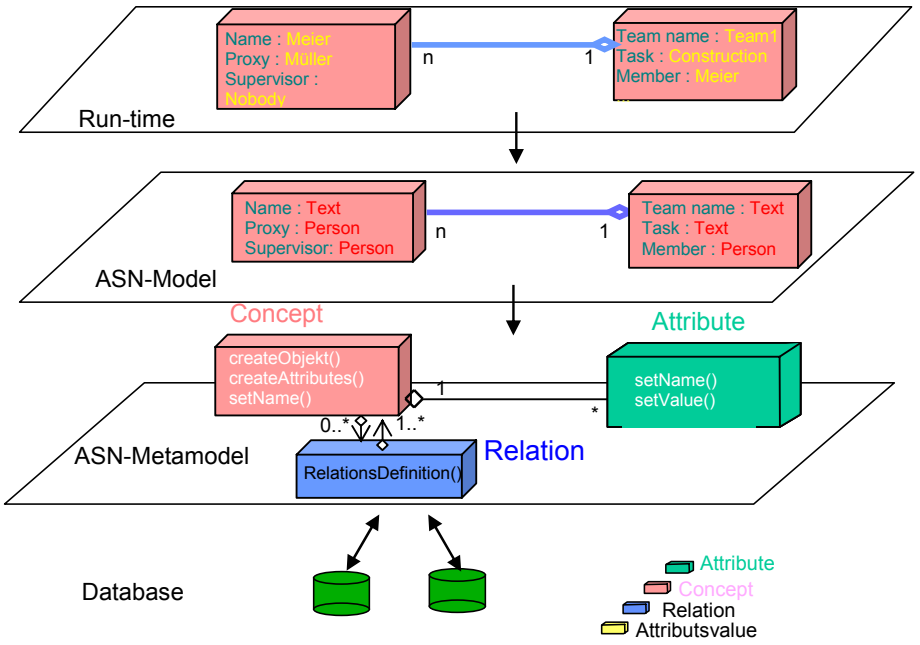

Figure 2. The ASN architecture

Further more, the ASN framework fulfils not only the modelling of various knowledge forms but also supports cooperation and online interaction between RPD users, synchronizes parallel access on ASN, realizes visualization of ASN, ensures the ASN consistence and manages versioning. The Active component is realized by the usage of constraints and ECA rules.

2.2 Knowledge Communication

Agent technology has proven itself to be a promising approach for implementation of frameworks for cooperative support [2]. The developed in the frame of Sfb374 agent framework [3], [4] was designed with the exclusive goal to facilitate RPD participants with provision services for knowledge retrieval, aggregation, monitoring and process coordination. The following requirements given by the RPD process and Agent concept are fulfilled.

An Agents system provides:

- Distributed services; agents can be instantiated and activated in different geographic locations. For example a user can choose the place for access to an information retrieval, and thus get use of the free resources of the network. He can query an agent located on the closest to him supporting host, or even locally. This approach assures scalability and prevents overloading of central access-point.
- Redundancy; the services provided by the agent system should be available at any time on demand. In case of failure of service or the hosting environment the failed functionality should be replaced by creation of the same agent type on another place. The user simply sends a query to an agent, referencing it with a name. If the agent does not exist it is created transparently for the user.
- Cooperation of agents; this is a preposition in the design of multi-agent networks where every agent can profit from the functionality of the others. In the case with RPD, an example is a user can give a complex question as a query which is separated by the agent to several elementary queries which are sent to the retrieval agent. Returned results are aggregated and delivered back to the user. An example of such case is the following

query: "Find from the existing in the system details such a combination to produce a car model with price under $10.000". This query is then sub-divided to the following sequence of queries expressed in pseudo-query language.

Example:

Q1 = "*what are the components of the concept car?*"
Q2 = "*iterate for every combination from Q1*"
Q2.1 = "*sum of components Q2 < 10 000*"
if true
Q2.2 = "*aggregate a car model with components from current Q2*"
return "*model from Q.2.2*"
else return "*empty result*"

Four basic agent types are specified to serve the RPD users:

- Retrieval; serves knowledge retrieval by a given query from the user. The specially developed language for accessing semantic network representations is called Active Semantic Network Query Language (ASNQL) [4]
- Aggregation; serves in finding new knowledge from the existing one with means of user questions.
- Coordination; directs the actions of multiple RPD participator according to a predefined process model
- Monitor; accepts as input conditions and returns queried properties values in the moment of satisfied condition

See Table 1 for more details about the implementation.

2.3 Process optimization

RPD was described as a continuous adaptation and exchange of accumulated knowledge during the whole process of development. As such, without a concrete model which supports this adaptation, there is a clear risk of process disbalance, resource locking and lack of synchronization in its sub-processes. In addition to this risk, the supporting software modules also suffer from need of automated management and adaptation to the needs of RPD. The research, which was done to figure out existing methods on process optimizations and safe adaptation in software supported systems outlined the base theories estimated to be appropriate as a reference for implementation of an adaptive model – Lehman's Theory of E-Type Systems [2], [5] and their evolution and secondly Stanford Beer's Viable Systems Model (VSM) [6], [7]. As Lehman showed, the processes in the business domain are strongly related with the specification, development and management of software for the support of the domain. A conclusion part from his E-type systems theory states that a software supported system can be represented as a multi-loop, multi-agent feedback system [8]. The requirements for the proper operation of the sub-processes in such systems are given by the VSM. Proposed and developed by Stanford Beer, it was initially designed for the purpose of understanding, predicting and controlling organizations of different nature – biological, social and enterprise. The VSM is based on studies and observation of the knowledge and information flow in many organizations over a period of thirty-years. Beer's goal was to discover the invariant structures and behaviour in open systems and describe them with cybernetic and control theory terms. Main properties of viable systems are adaptation and stability in time. Without these properties an open system may become unusable to accomplish the goals it had been built for. It is however an unbearable task to adopt such a global model of organization and

apply it to an already developed and ongoing complex process as RPD. A single step migration would imply too many changes (especially keeping the semantics inside the initial organization while translating them) and is therefore too risky and error-prone. That is why as a starting step we aimed to adopt the VSM communication requirements and namely the requirement for communication channel requisite variety, a rule which assures proper communication between elements and groups as well as potentiality for overall system adaptation. The second step is to put the base for extension of the model that is stored in the ASN RPD with the VSM-like self-reference model for functional grouping. These two added features allow user queries to the knowledge base to be evaluated better on the base of functional membership and position in the model's recursive structure. In the following paragraphs are presented briefly the properties and requirements of VSM and the Active Semantic Network Extension for adaptation of RPD knowledge for support of viable processes.

2.3.1 VSM Properties and Requirements

There is extensive literature on the VSM and its application in knowledge management. However there are few important things to be mentioned. The distinct point in the cybernetic theory of organisations is that viable systems are defined as recursive self-referencing systems that contain other viable systems. These can be then modelled using an identical cybernetic description as the higher levels in the containment hierarchy. Beer expresses this property of viable systems as cybernetic isomorphism.

Principles of the model:

- Self-reference; a term to describe how each part in a system makes sense in terms of the other parts. The system defines or produces itself based on the parts and their arrangement. This property is also called logical closure and is related to identity, self-awareness, self-repair and recursion itself.
- Homeostasis; biological term representing the main characteristic of live structures - maintenance of critical variables within certain limits to ensure stability of a system in order to react to changes in the environment

Requirements:

- Variety requirement; the Law of Requisite variety as a critical and important cybernetic rule which has to be kept for guaranteed system viability
- Dynamic requirement; the channels connecting the controlling and execution groups should allow higher transfer capacity at a given moment than the originating subsystem can generate at the same moment
- Transitional requirement; whenever transmitted variety passes a border of transduction, it is necessary for the transducer to have at least the capacity of the communication channel’s variety.

2.3.1.1 RPD and Software Variety

The requirements given by VSM apply to the cybernetic definition of variety: *the number of states a system may have in its state space*. As a typical elementary example from the real life one can imagine a thermostat that controls the temperature in a room. There are two types of variety in this system – disturbance (environment temperature) and regulating (thermostat’s control range). However, this definition for variety is not directly applicable to the nature of software because of the qualitative characteristics of the different software elements included in the supporting system: communication protocols, XML message schemas, remote

interfaces, component descriptors, event logs, etc. On the other side the variety in the process of RPD variety is seen as the different ways of communication and information exchange – different versions of documents, plans, analysis, visual and audio feedback. We call this type of variety ***observable variety***. It is a *qualitative measure for the variety which a certain system element may introduce in the process of communication with other elements.* Variety in ASN is expressed as the set concepts relating to the act of communication establishment (see section 2.4).

2.3.2 Existing implications of VSM

VSM is known for its application in organization and management of companies and environmental management and planning [9]. However, the focus of this paper is on its application in the symbiosis of adaptive processes and their supporting software. Such combination requires a look at the existing project management methods and software design efforts which include VSM-based models. Relevant to the field of project management is the work of Schweininger [10] which exposes the values of cybernetic approaches and recursive structures in management of complex projects. Extensive research on VSM usage in software components has been done by Herring [11], [12], where the VSM is used as a reference model for specification of component interfaces for integration in complex software systems. The research also proposes methodology for development and is a valuable guide for design of viable components interfaces. However it does not touch the subject of process definition and knowledge communication and flow mechanisms which a viable systems necessarily includes. Being aware of these approaches and aiming to add the missing bridge to a practical VSM-based realization we focused on RPD knowledge storage, process representation on different system levels and its internal handling by the software runtime.

2.3.3 ASN Autonomic Extension Model

As discussed, the primary aim of applying the VSM to the existing ASN oriented design is related to the knowledge retrieval experience - optimization of retrieval results and better user navigation. In order to keep compatibility with the current applications working with ASN and in order not to change the meta-model it was decided all components of the extension to be built using inherited ASN Concepts. Four extensions were added – Autonomic Element, Autonomic Manager, Managed Communication Channel, and VSM Functional Group. The adjective “Autonomic” was chosen because of its direct relevance to the currently ongoing research initiatives Autonomic Computing and Autonomic Communication to which our research team is committed.

On Figure 3 is shown a reduced version of the ASN meta-model and the extension model with the help of which is realized a recursive structure with managed communication channels. As seen, the three interacting components *AManager*, *CommunicationChannel* and *FuncGroup* share the same root concept – *AElement*. This is needed in order to fulfill the requirement of self-reference and self-containment, provided by *FuncGroup*. An instantiation of *AElement* produces an object the properties of which can be monitored or modified by external control units through a common interface, forming an element known in Autonomic Computing literature as the term *Managed Element*. The *AManager* is the component responsible for the monitoring and acting on managed elements and its primary purpose is the acknowledgment for completed requisite variety in the *CommunicationChannel*. The concrete sequence of steps for establishing a managed communication channel is discussed in Section 2.4**.** A viable system contains by definition units which are themselves viable systems, and namely, the group responsible for operations (the lowest system level) forms a new subsystem which possesses the properties of the higher system level to which it belongs. The *FuncGroup*

concept represents this recursive structure expressed on the diagram as aggregation of recursive groups and managed elements. The additional constraints, such as properties of the groups depending on the level number are left to the concrete implementation of the model in the runtime module.

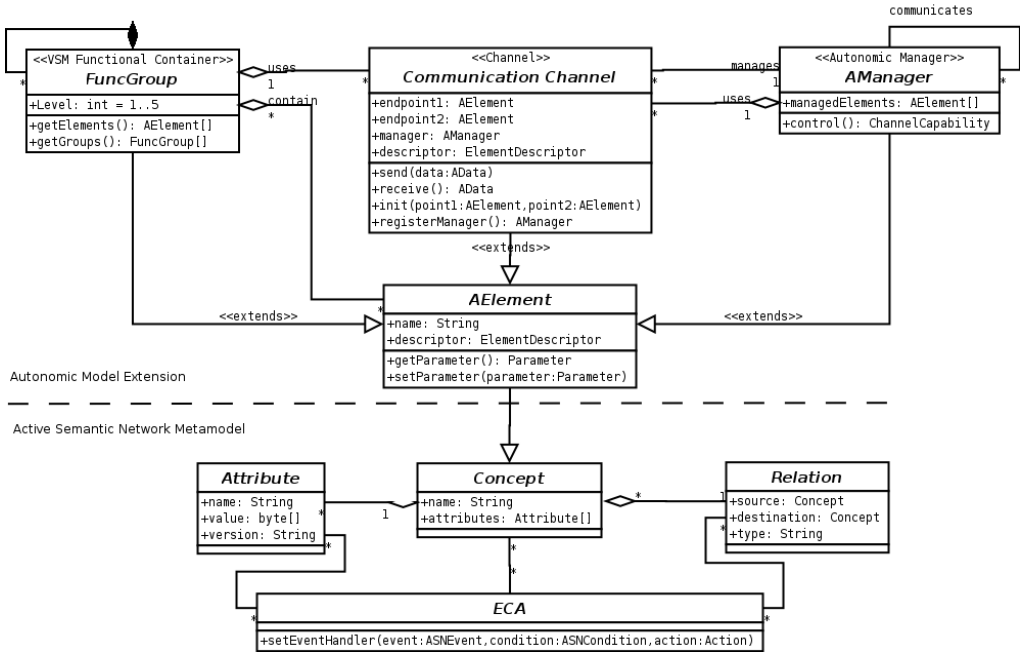

Figure 3. ASN Meta-model with Autonomic Extension

2.3.4 Adoption methods

The ASN Extension provides the needed base for the expression of semantics inside a viable system, but without existing relations with the rest of the RPD model it is useless. The agents and their decision algorithms for retrieval need these relations to evaluate the queries and control the process. As pointed out, a gradual adoption is needed and there two phases in which this can take place – direct assignment in the phase of knowledge addition and explicit inheritance of existing concepts. The two methods allow to be used together throughout the RPD process without affecting it.

- Direct assignment through addition – explicit specification of relations in the process of adding new objects. Objects that are new for the process are assigned to functional groups at the moment of their instantiation in the RPD support system, before they were actually stored in the knowledge base (Figure 5). This requires the sub-process in which the object acts to be already aware and adapted to act on the basis of communicating functional groups. This way the communication channels for exchange of knowledge between groups have the needed requirements to be created and operate.
- Explicit inheritance – relations between groups in VSM are expressed by the existing communication channels between them. Because of the absence of such in the initial RPD model, a set of concepts and relations are modified to inherit the properties of AElement and CommunicationChannel concept respectively. The inherited concepts are still compatible with the former model and can be used without affecting the process.

2.4 Control of RPD Knowledge flow

The information between actors in the RPD process is exchanged in different ways – verbal, visual or audio, with paper documents, electronic documents, etc. However, knowledge representation and communication are not the only requirements for the aim to reach qualitative process support. The supporting module needs an adequate knowledge flow control to assure that the supported processes will remain consistent as required by the above stated model requirements. An example for the relations in a system for support of any operational domain is showed on Figure 4. This is Lehman's model which illustrates that the operational domain and its support are instances of one and the same specification. There is a need of validation and compatibility check between them to assure proper satisfying quality of support.

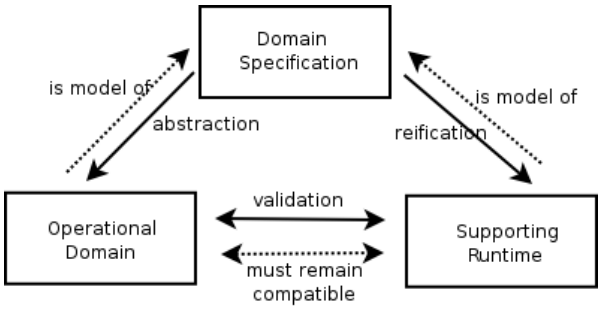

Figure 4. Dependencies between operational domain, domain representation and supporting runtime

Extremely important requirement is the runtime to have the means to handle in a flexible way the changes in the domain specification. A change and its improper handling may result to faulty relations or inability for knowledge transfer between communicating parts. Following the model on Figure 4 and applied to the concept of knowledge representation in RPD environment results to the knowledge flow diagram on Figure 5. The RPD participant is part of the support domain (RPD) and interacts with the supporting environment which internally represents the process on two levels – ASN knowledge and runtime objects.

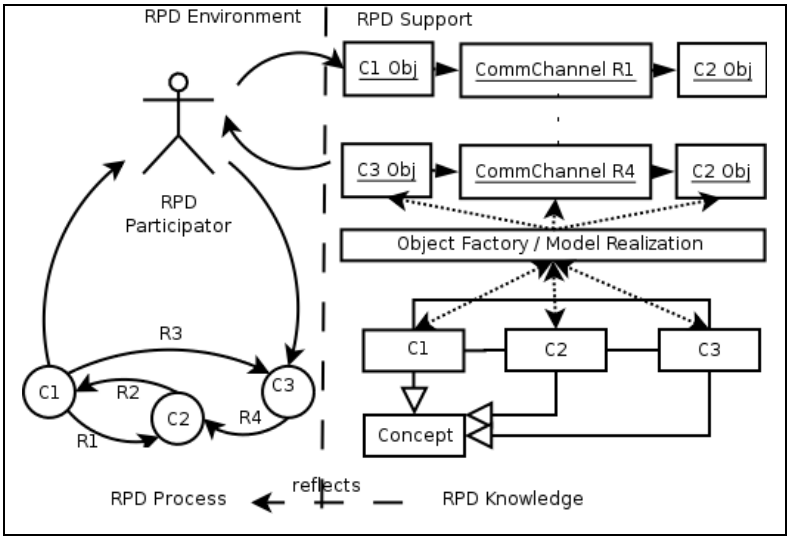

Figure 5. RPD Knowledge Flow and Model Realization

The ASN Knowledge reflects the real RPD process which includes the formally named communicating objects C1-C3 and their communication channels R1-R4. These are represented as ASN Concepts. The object factory is a bridging mechanism that takes care of the instances of the concepts and communication channels with the properties from the knowledge base. These objects are used by the execution logic, which was omitted to simplify the diagram. The instances serving communication (R1-R4) between the objects have an important role in the control of the transmitted informational variety. Practically the RPD communication capabilities are verified on request for channel establishment.

For a successful communication between the two endpoints A and B, a channel connecting them is checked for provided and handled variety and return one of the possible states it can have:

- Both endpoints are compatible and will have accurate two way communication
- Both endpoints are incompatible and will not be able to transmit accurately variety
- Endpoint A will receive variety accurately, but B not (A to B accurate).
- Endpoint B will receive variety accurately, but A not (B to A accurate).

The concrete sequence of interaction of objects with the communication channel is shown on Figure 6.

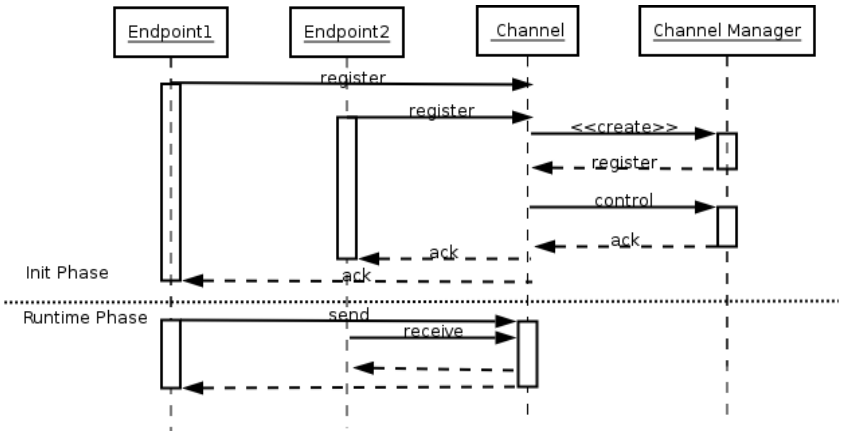

Figure 6. Communication channel and Channel Manager interaction

2.5 Benefits

The discussed approach has several advantages in comparison with the traditional methods for reliable software development [13, 14].

- Proposes a clear and polished organizational model for support of RPD
- Controls distributed system entities' interaction in sense of their compatibility with the goals of the system
- Monitors overall ability of the system to be adapted in time, without interrupting the business process which uses it
- Inconsistency problems are resolved before they caused cascaded failure

3 Realization

Since the beginning of the development of the components for the discussed model it was decided that an implementation should use as much as possible open standards and technologies. This allows interoperability with other systems and easier support when it comes to configuration, management and updates of the system. Here are described the used standards and methods for every of the levels in the architecture. The system is designed to be distributed and scalable and uses middle-ware components which offer the advantages of additional separation of the services and implementation of routine problems like object persistence. As a programming language was used Sun Microsystems Java 2 and its specifications Java 2 Standard Edition for the implementation of the Agents Framework and for the ASN run-time the Java 2 Enterprise Edition. Concrete information about the implementation of the layers is provided in Table 1.

Table 1. Used standards and technologies

Layer	***Standard/Technology***	***Reason***
Data Persistence	SQL, MySQL	Widely adopted, fast SELECT results
ASN Runtime	Java 2 Enterprise Edition, JBOSS Application Server	Platform-independent, Enterprise standard, Automated persistence
ASN Meta-model Components	Enterprise JavaBeans with Container Managed Persistence	Component-oriented design, database independent
Agent Framework	Java 2 Standard Edition	See ASN Runtime
Agent Communication Protocol	Token-Ring-like based on XML messages	Open standards, proven stability and operability
Agent Network Layer	Multicast, Reliable Lightweight Multicast Protocol (LRMP)	Group-oriented, supported by enterprise vendors

Examples:

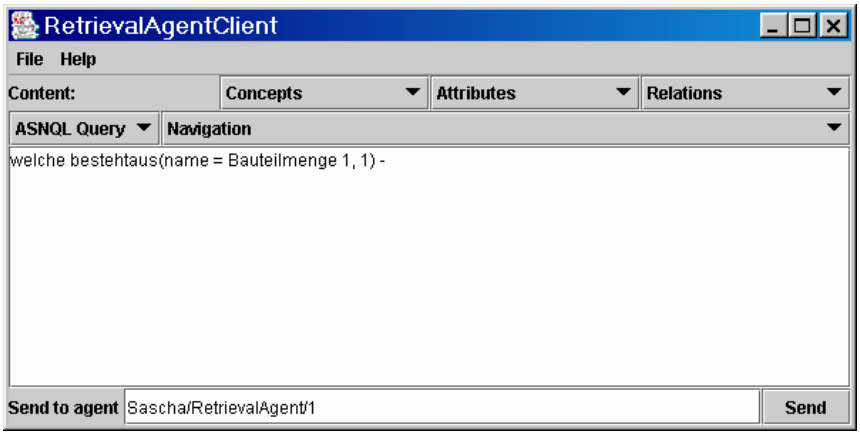

Figure 7. Example for ASNQL User Query to a retrieval agent

Figure 8. Example for a query result

4 Conclusion

Based on principles of Cybernetics and Autonomic Computing we designed a software architecture for the purposes of RPD. Key properties of ASN and MAS as middleware were discussed in relation to their organization to support a viable RDP knowledge integration and communication. Key advantages of this approach are optimization of agent functionality, stability and clear functional separation of the RPD knowledge.

References

[1] Roller, D., Eck, O., Dalakakis, S., "A Knowledge based support of Rapid Product Development" Journal of Engineering Design, Taylor & Francis Ltd, Vol. 15, No. 4, 2004, pp. 367 – 388.

[2] Ye, Yeming, Churchill, E., “Agent Supported Cooperative Work”, Series Multi-agent Systems, Artificial Societies and Simulated Organizations, Vol.8 , Springer, 2003

[3] Diederich, M. K., Leyh, J., “Agent Based Middleware to Coordinate Distributed Development Teams in the Rapid Product Development”, Proceedings for the Conference on Product Development, 1st Automotive and Transportation Technology Congress and Exhibition, Barcelona 2001.

[4] Dalakakis, S., Stoyanov, E., Roller, D., “A Retrieval Agent Architecture for Rapid Product Development”, Perspectives from Europe and Asia on Engineering Design and Manufacture, X.-T. Yan, Ch-Y. Jiang, N. P. Juster, eds., Kluwer Academic Publishers, 2004, pp. 41-58.

[5] Lehman, M. M., with DE Perry and JF Ramil, “On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution” Proceedings Metrics'98, Bethesda, Maryland, 1998.

[6] Beer, Stafford, “The Heart of Enterprise (The Managerial Cybernetics of Organization”, John Wiley & Sons, ISBN: 0471275999, 1979.

[7] Beer, Stafford, “Brain of the Firm (Classic Beer Series)”, John Wiley & Sons; 2 edition, ISBN: 047194839X, 1994.

[8] M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution ," Proceedings of the IEEE 68(9), pp. 1060—1076, September 1980

[9] Klark, M. “A Cybernetic approach to Sustainable Development”. Planning in North West England during the 90s, GBER Vol. 2. No 1 pp 34-39.

[10] Schwaninger M., Koerner M., “Managing Complex Development Projects: A Systemic Toolkit Based on St. Gall Management Framework”, Discussion Paper, No 37, 2000.

[11] Herring, C., Kaplan, S., “The Viable System Model for Software”, 4th World Multiconference on Systemics, Cybernetics and Informatics, SCI2000, 2000.

[12] Herring, C., Kaplan, S., “Viable Systems: The Control Paradigm for Software Architecture”, Australian Software Engineering Conference, 2000. Canberra.

[13] Rothermel, G., Harrold, M., “Analyzing regression test selection techniques”, IEEE Transactions on Software Engineering, 22(8):529-551, August 1996

[14] Rothermel G., Harrold, M., “A Safe Efficient Regression Test Selection Technique”, ACM Transactions on Software Engineering and Methodology, 6(2): 173-210, April 1997