Refining partial systems applying architectural view patterns

Mario Kupries
College of Information Systems
Zayed University
United Arab Emirates
Mario.Kupries@zu.ac.ae

Erika Horn
Institute for Informatics
University of Potsdam
Germany
ehorn@soft.cs.uni-potsdam.de

Abstract

Subject of this paper is the derivation of architectural views assigning a building block character and semantic status to components, connections and systems. At first, the concept of architecture types will be introduced. Next, architectural frames are derived as common development frameworks for application systems. The functional grouping of components and connections in architectural frames leads to the identification of partial systems. To substantiate the partial systems a methodical approach is introduced for their analysis and classification resulting in architectural view patterns. The application of a special architectural view pattern will be described and discussed for interactions in multi-agent systems.

Keywords: Architecture modeling, object-orientation, patterns, connectors

1 Introduction

Latest software application systems are characterized by large complexity, openness, heterogeneity, etc. and necessitate substantiated engineering approaches for their analysis, modeling, implementation, validation and adaptation. In this article the view is taken that for the engineering development of latest software application systems their building elements have to be structured in logical formations and set into relations to each other. As common in engineering disciplines and as a technical approach to systems, software architecture theory (e.g. [14]) is applied to and interpreted for software application systems aiming at a building element technology and abstraction levels. Software architectures will be described in their structural [14] as well as operational properties and specifics [5]. Since software application systems are to be developed into architectural frames, their partial systems and building elements have to be classified and substantiated. Therefore architectural views are introduced and used by applying architectural view patterns. The pattern serve the enlightening of application techniques and semantics of partial systems and their building elements. The use of architectural view patterns results in substantiated templates programmers can make use of.

2 Applying software architecture theory

The modeling of software architecture takes place on a high level of abstraction and is abstracted from implementation details as well as from concrete application cases [15]. On this level of abstraction the common software architecture theory describes the structural formation of software systems as collections of execution units (components) together with their properties and as collections of dependencies and relations (connectors) together with their rules ([1], [15]). Thus, the software architecture is a mental, time variant and technical artifact which describes software application systems in a model-based manner and according to a specific level of abstraction.

To classify architectural views in software architecture theory, architecture types and architectural frames will be introduced at first. Based on this introduction partial systems are identified and derived. For their substantiation architectural view patterns will be applied to in chapter no. 3. Therefore, an approach for the construction of basic architecture view patterns is being described. The concept of architecture view patterns will be used for a substantiated enlightening of interactions in multi-agent systems throughout chapter no. 4. The article comprises of class hierarchy models, technical diagrams and code fragments.

2.1 Architecture types

An architecture type describes the abstraction of properties of components and connectors as well as their composition into systems. Despite the structural formation architecture types describe the operational treatment of software systems like the construction, installation, adaptation, etc. In figure no. 1 classes are introduced for the architecture theory vocabulary [15] and set into relations to each other applying concepts of object-orientation. This results in an abstracted modeling of a basic architecture type for software application systems [6]. Using this approach abstractions and compositions of reusable building elements are enabled. On a meta level analysis, abstractions, compositions and ascertainments of architecture types are enabled again.

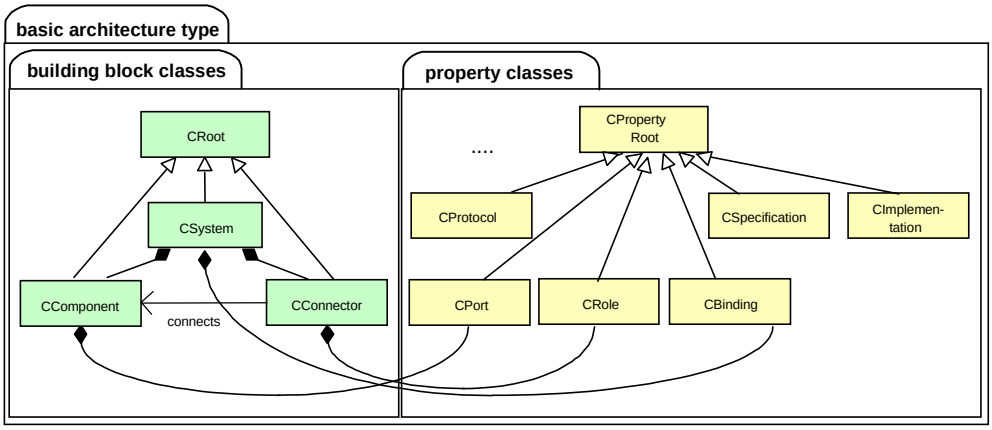

Figure 1. Basics architecture type for software application systems [6]

Of important relevance in figure no. 1 is the assignment of an explicit semantic status and application techniques to the execution units, connections and their compositions into systems. In this approach components, connectors and systems are modeled as explicit building elements owning properties like interconnection points (port, role), protocols, security levels, implementations, etc. In figure no 1. properties and building elements are logically grouped into two areas. The extensional abstractions of common properties leads to the creation of the equivalence classes '`CRoot`' and '`CPropertyRoot`'. Applying object-oriented modeling of building elements architecture types are prepared application-technically and architectural frames are derived. Architectural frames represent abstracted knowledge of software developers [9]. The position taken here is that software application systems are to developed into architectural frames [8].

2.2 Architectural frames

By using abstractions and applying object-oriented principles hierarchically ordered architectural frames for classes of software application systems are derivable [8]. An architectural frame has a descriptive character. As a pattern (in terms of [2]) the framework serves as a template which has to be interpreted for applications. An architectural frame is derived from an architecture type. Architecture frames serve

- identification, abstraction, classification, description of application-technical building elements
- informal description of semantics
- specialization of building elements
- prefabrication of re-useable building elements
- composition and re-use of building elements
- refinement of partial systems and building elements
- requirement survey for platforms and their services.

For a class of software application systems an architectural frame describes building elements and their relations on an instance level. Figure no. 2 represents the basic architectural frame for software application systems.

According to [9] the basic architectural frame for software application systems consists of the partial systems user interface, application domain, component-specific services and external resources. The *user interface* consists of interface components and connectors for users to input and hand over their tasks and preferences as well as for administrators. In case of missing parameters, uncompleted knowledge bases, etc. supporting information will be requested by the system from users or administrators using special assistants. Software application systems make use of existing data and services integrating legacy applications. The wrapping of databases, applications, file systems or network connections is taken over by special *resource* components and connectors. Basis machines and run time environments are used by software application systems as special *services* provided by a platform. The spectrum of services comprises of life cycle functions to create, migrate or terminate components and connectors, of directory services, of knowledge and language registers, etc. The partial system of *application domain* consists of specific and derivable components and connectors according to the application semantics. Management components take over the allocation and delegation of partial tasks to and the monitoring of task components using special connectors. Task components supply output in the sense of processing complex orders (e.g. search, analyze, evaluate).

The grouped partial systems of the basic architectural frame are derived by logically ordering of their functionalities. For their engineering substantiation special inspections – views – are applied to.

The substantiation of their properties and specifics is taken place by model-based views onto their structural formations and operational treatments.

2.3 Views and partial system refinement

The concepts of separating, deriving and using architectural views are not unique. In [4] the search of architectural commonalities and principles from practice led to a loose coupling between code, module, execution and conceptual views. In [11] the five concurrent views logical, process, physical and development view are introduced as an architectural representation.

In this article we are taking the position that due to the complexity of software application systems' functionalities, documentations, platforms and technologies their model-based decomposition into partial systems offers view on their structures and behaviors in common. The model-based inspection is always derived under a specific aspect. At every moment of inspection, special properties are relevant and partial systems are delimited which have relations to each other. In the model-based treatment the partial systems are ordered topologically. Both, adjacent as well as remote partial systems can be interrelated [5].

The set of partial systems representing architectural views is not fixed, but comprises of special views according to the specifics of a software application system. It is task of research and practice to identify, analyze, model and prefabricate a standard set of architectural views common to all software-intensive systems. To this set task view, information view, organizational view, resource view, time/schedule view, etc. can belong to. The use of such standard components, connections and (partial) systems can be integrated in all abstraction and hierarchy levels of architectural representation.

In this article, models of partial systems represent informal architectural views. Different modeling techniques, e.g. UML class diagrams, state transition flows, code fragments, are applied and used. If there exist more than one partial model of an architecture and the models are dependent (for example not orthogonal) then an integration model is necessary.

The abstracted knowledge on partial systems will be compiled in complex catalogues, called architectural patterns. For classes of software application systems architectural patterns can be prefabricated and thus serve as templates. They can be seen as an extension of architecture description languages.

3 Deriving architectural view patterns

As engineering inspections architectural views are applied to specify structure, behavior, application semantics and techniques of building elements. Due to abstraction and recursion building elements are again treated as systems.

According to the embedded building elements of an architecture type and their mapping onto an architectural frames application-independent and application-specific views can be identified. Application-independent views are common for a class of software application systems. Examples of application-independent views are integration of legacy systems, human computer interaction, service integration. Application-specific views are mainly applied to the application case and domain.

3.1 Constructing architectural views

The construction of architectural views is an engineering process resulting in architectural view patterns. At first, the class of software application systems has to be determined. An architectural view pattern is a derivation of a partial system according to a specific aspect. As introduced earlier the partial systems are assigned an explicit building element character.

The process of constructing architectural view patterns is related to architectural viewpoints [10] defining concerns and resources to create architectural views.

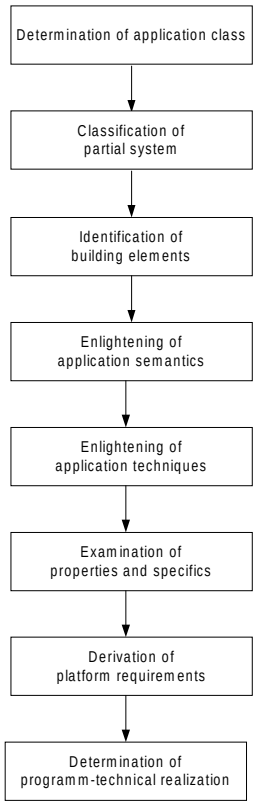

Figure 3. Steps constructing an architectural view

Commonly, the building elements component, connector and system are described and modeled. They are specialized according to the class of software application system they belong to. Assigning the building elements to an application class the enlightening of their application semantics and techniques has to be taken place.

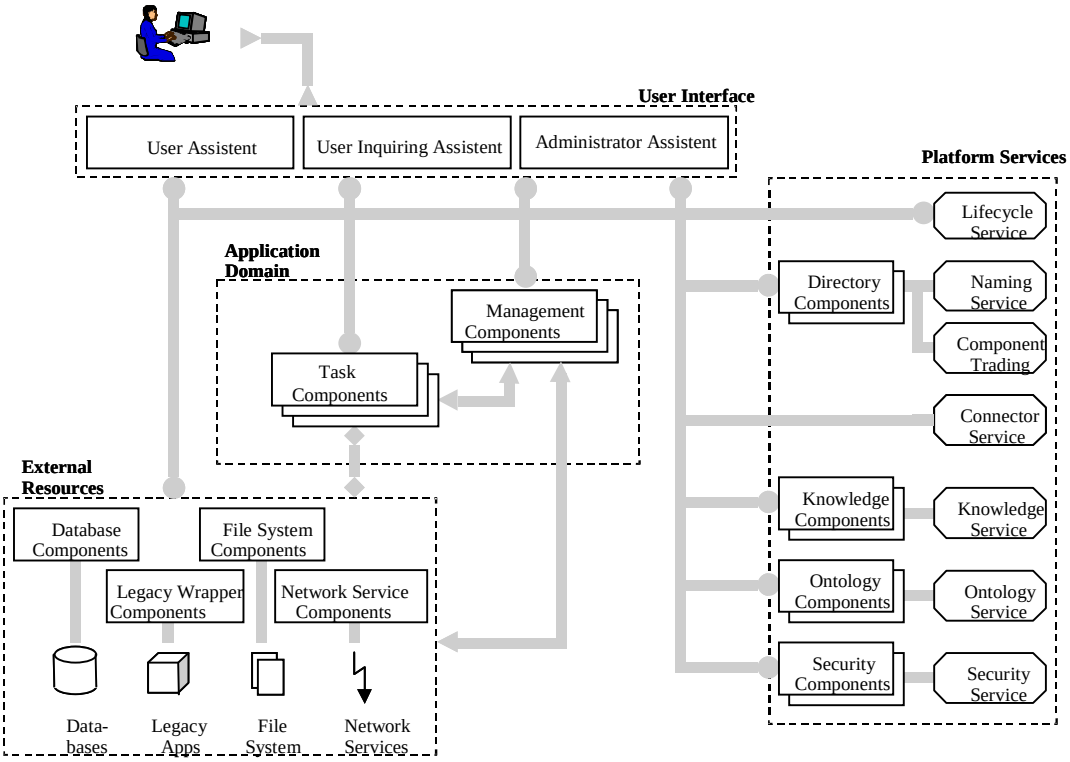

Figure 2. Basic architecture frame for software application systems

Regarding their application-technical context the zooming into their software-technical properties and specifics is enabled. The information derived from the engineering examination of partial systems leads to the compilation of details onto platform services as well as onto the program-technical realization. Figure no. 3 represents the steps constructing an architectural view. The results of architectural views are stored into architectural view pattern (table no. 1). As outlined in [3] the pattern has to meet the IEEE 1741 [10] requirements.

Architectural view patterns serve the substantiation of frames' partial systems and lead to the specification of building elements and their relations to each other of a partial system. Requirements onto platform services as well as templates for the program-technical realization are compiled in architectural view patterns. Thus, architectural view patterns are complex catalogues comprising of abstracted and reusable knowledge. Each slot of an architectural view pattern hosts complex information and is represented again by patterns and models. The pattern slots are not fixed since special patterns can be derived and prefabricated for special software application systems.

An architectural view is specific for a class of software application systems. An architectural view is a technical description and specification of an building element together with its striking techniques, semantics and properties. For architectural views state transition models, life cycle models, etc. can be specified.

3.2 Compiling architectural view patterns

According to the class of software application system and its domain special view patterns can be derived from the basic architectural view pattern represented by table 1. Applying object-oriented principles concepts of reuse, abstraction, composition, etc. of architectural view patterns are enabled.

Having introduced the concept of architectural view patterns the next paragraph will explain the application of this concept for substantiating interactions in multi-agent systems.

Table 1. Basic architectural view pattern

Pattern: PArchitecturalView	Pattern name pre-fixed by a 'P'.
Class of software application system:	Description of the specifics and domain of the software application system.
Partial system:	Delimitation and classification of the concerns and architectural issues.
Building element:	Classification of the building element (component, connector, system). **(Note: Due to recursion a system is being abstracted into a component.)**
Stakeholders:	Identification of responsible specialists and their audience as well as their duties.
Application techniques:	Analysis and modeling of special building elements and processes, e.g. for task execution, services, platforms. Special model types are used.
Application semantics:	Analysis and modeling of special building elements and processes of a domain according to rules, organizational structures, etc. Special model types are used.
Properties and specifics:	Specification of structure and behavior. Description of common and special software-technical properties, e.g. security, mobility, interoperability. Specification of structure and behavior. Special model types are used.
Platform services	Requirements onto functionalities of platform services.
Program technical realization:	Preparation of a template.

4 Architectural view on mas interactions

Multi-agent systems (mas) provide new approaches to the interaction, take-over and execution of tasks. With their characteristic features, such as autonomy, mobility or the ability to negotiate, they make great demands on analysis, modeling and development and at the same time open up new fields of application. The relations between agents are a peculiarity in multi-agent systems. They have to be complex, variant and negotiable. The interactions between agents are granted an explicit modeling status. Building blocks for interactions in multi-agent systems which can be identified as application-specific interagent connectors are abstracted, modeled and prefabricated. By applying of architectural view patterns the demands on interagent connections, the analysis and modeling of classes of interagent connectors and their matching assignment to special agents are elucidated.

4.1 The architectural view on interconnections

4.1.1 Determination of application class

A software *agent* is a computer program working autonomously and goal-orientedly in the name of an authority [14]. An agent has got its own execution profile and is able to execute tasks on its own initiative and cooperating with other agents during its life cycle. *Interagent connectors* serve the communication and interaction between agents in an agent system. Interagent connectors provide protocols regulating the interactions between agents, enabling the use of the functionality of other agents or allowing access to external resources [7]. An *agent system* contains agents and interagent connectors and manages the relations between them. It provides operations that allow the generation, interpretation, execution, termination of agents and enables the establishment of connections between agents and authorities as well as between the agents themselves [7]. Based on the object-oriented analysis and modeling of agents, interagent connectors and agent systems, complex architecture types and have been emerged and architectural frames where derived from (e.g. [6], [7]).

4.1.2 Delimitation of partial system

Figure no. 2 outlines instances of building element classes allocating them to partial systems of the architectural frame. Hierarchies of architectural frames can be derived corresponding to architecture types. Along these hierarchies an increasing application-specific context of building elements is identified [9].

In this article we are going to apply architectural inspection for the partial system of service integration. Services are based on application-independent building blocks and concepts. They are derived and developed by examining, analyzing and abstracting application-technical requirements on platforms. Application-technical building blocks for interagent connections will be identified, classified and prefabricated. Composite building blocks are structured into and build of singular elements. The application of object-oriented principles enhances this approach [6]. In contrast to application domain, the refinement of platform services is not based on the allocation of prefabricated building element classes and their mapping as application-specific building blocks .

4.1.3 Classifying the building elements

The architecture-based approach assigns a building block character to interactions in software application systems. Next, we are going to focus onto the building elements called interagent connectors.

As special classes of interagent connectors e.g. `CIAInvocationConnector`, `CIATaskConnector`, `CIAInterfaceConnector` have been analyzed, modeled and prefabricated [13]. Interagent connectors for service integration are application-independent.

4.1.4 Determination of the application semantics

The interagent connection is a special interconnection type and is based on special classes of interagent connectors. In the behavioral assignment interagent connectors specify the cooperation of agents for the take-over and processing of tasks. Therefore the local behavior *p* (*p* for port) of the agent *ra* (*ra* for requesting agent) must be assigned correctly to its corresponding interconnection party p_{pa} (pa for providing agent). This is realized by a consistent and coherent merging of their behavior into an interagent connector coupling protocol $cp_{ra,pa}$. Figure no. 4 represents the behavioral assignment of corresponding interconnection parties in an abstracted manner. It represents a complex process assigning providing to requesting behaviors. An extension of this concept is represented by figure no. 5.

Due to their characteristics the agents treat interagent connectors as explicit building elements. Therefore, the interagent connectors have to fulfill specifics like non-deterministic connections, heterogeneity of participating building elements, volatility of connection, dynamic behavioral assignment, laterality of connections, etc.

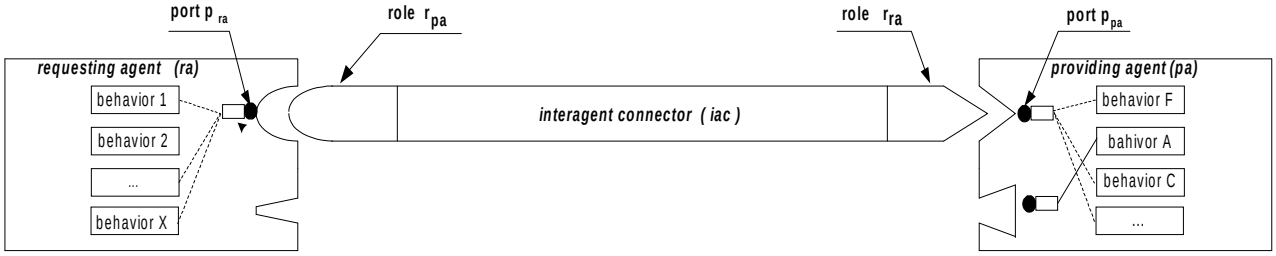

Figure 4. Behavioral assignment to corresponding interconnection parties

4.1.5 Modeling of the application techniques

Due to its specifics the interagent connection is based on several phases during its state transition. The phases of interagent connection describes the assignment of corresponding behavior according to syntactical and semantic correctness.

This assignment is traversing several phases wherein the interagent connector will adopt different stati. The phases of interagent connection comprise of construction, local and global matching, tuning of assignment, interaction of corresponding parties, destruction. The sequence and application of interconnection phases is represented by figure no 6.

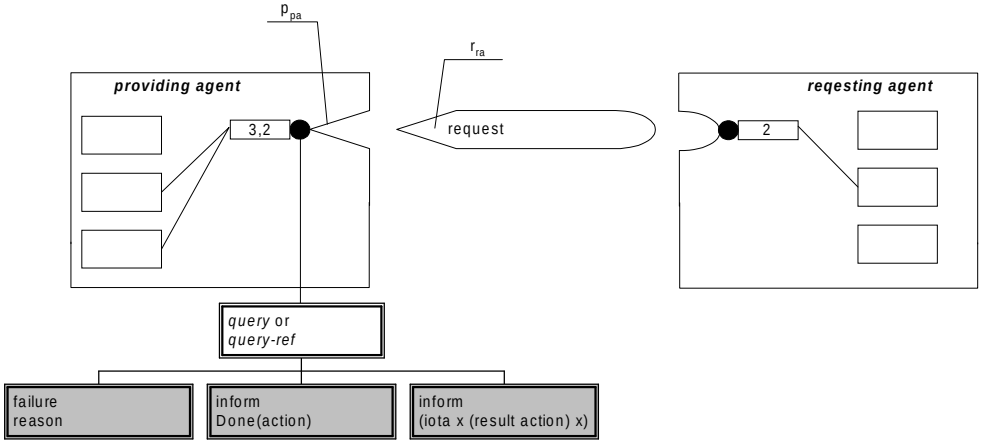

Figure 5. Behavioral assignment as intra-protocol selection

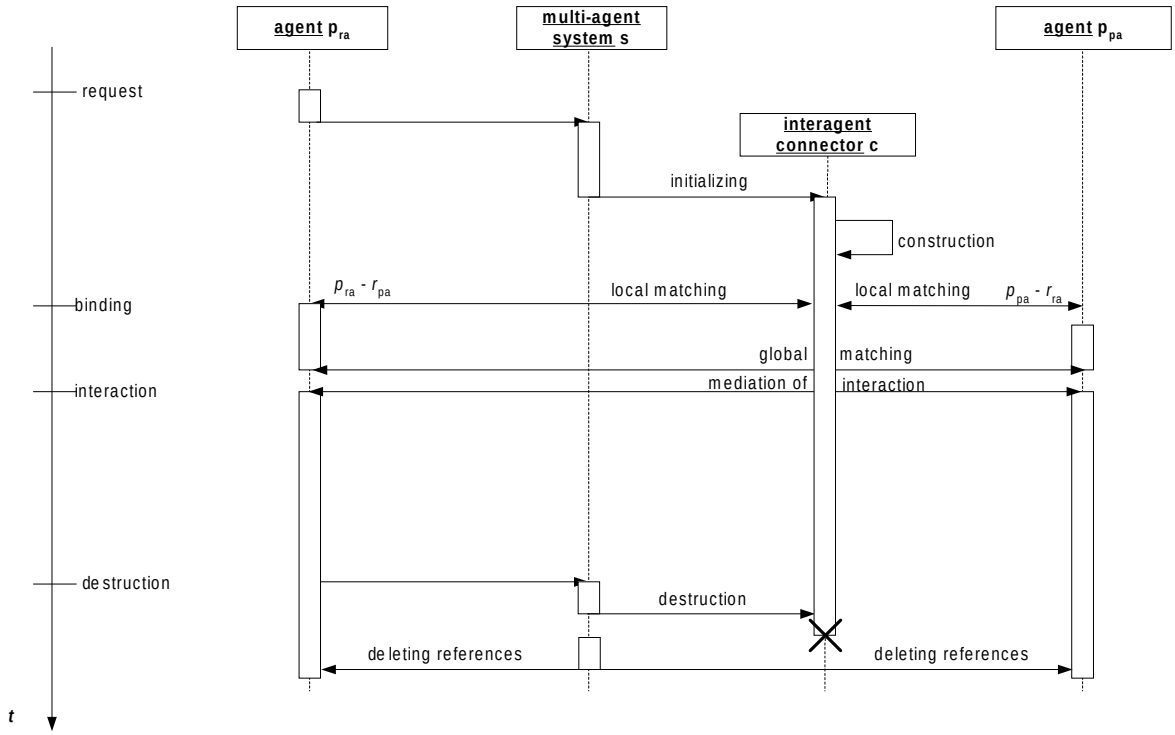

Figure 6. Sequence and application of interconnection phases

4.1.6 Modeling of the building element

Compiling of the information from the engineering substantiation of interagent connections, a basic model type [5] of interagent connectors can be given [13]. The class `CInterAgentConnector` is created by the composition of special interface, state, environment, protocol and language classes.

The represented class model in figure no. 7 serves the program-technical realization. Prefabricating, the special classes `CProactiveRole`, `CReactiveRole` and `CSensingRole` are derived as special profiles for interagent connectors. Figure no. 6 represents the model type of an basic interagent connector.

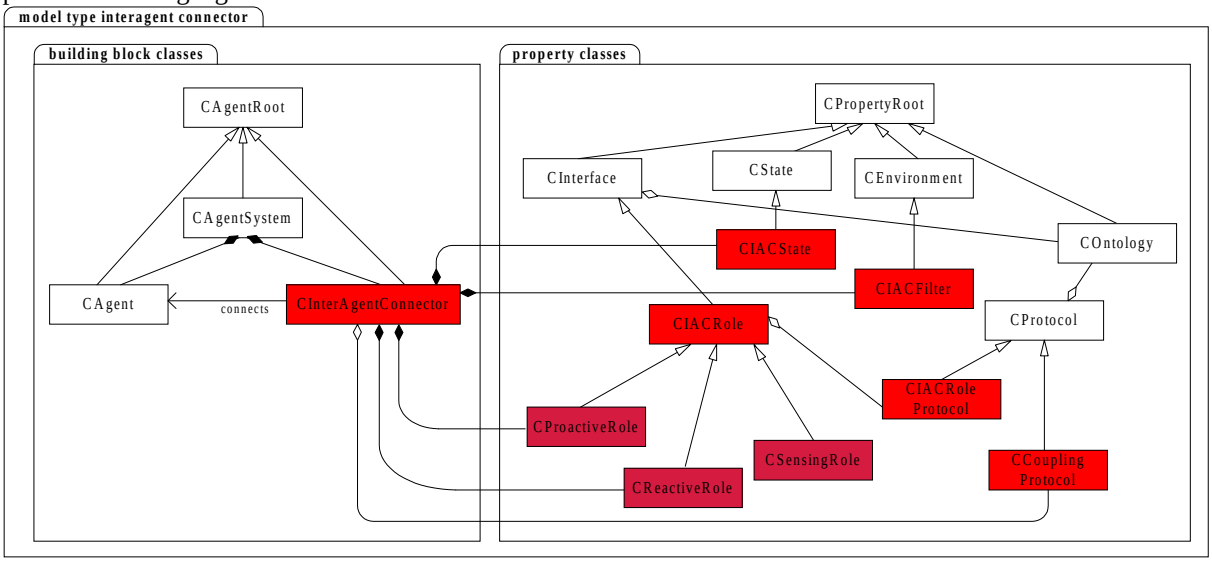

Figure 7. Model type of an basic interagent connector

4.1.7 Requirements on platform services

The program-technical realization and execution of connections prerequisites special services oriented towards concepts and models of the architecture-based approach and meeting the requirements of application-specific building blocks. Therefore, requirements for the partial systems will be determined. These requirements are according to [13]

- agents, connectors as application-technical blocks
- prefabrication and re-use of agents and connectors
- analysis, classification, composition of agents and connectors
- special services for separated management, assignment of agents and connectors
- dynamic management and assignment of connectors
- enabling primitive, complex behavioral assignments
- management of phases and states of task execution and interagent connections
- syntactical and semantic matching
- information as to properties, state, type, history, etc, on agents and connectors.

The identification, analysis, abstraction, modeling and prefabrication of classes of interagent connectors was done on the level of architecture type modeling and thus independently on implementation details and abstracted from concrete application cases. To implement and map interagent connectors as separated, explicit and prefabricated building blocks with own semantic status onto services of platforms, it has to be substantiated and refined as a technical model. The refined service structure has to meet the requirements of the architecture type-based approach, of application-technical interagent connectors and phases of the common processing model [12].

4.1.8 Program-technical realization

Code no. 1 represents the IDL specification of a common class of an interagent connector. It specifies the composite structure of figure no. 7 and its operational treatment.

```
(1)     // ----------------------------------------
(2)     // cinteragentconnector.idl
(3)     //
(4)     // Class for basic interagent connector
(5)     Interagentenkonnektor
(6)     //
(7)     //    M. Kupries
(8)     //    Solaris 2.5
(9)     // ----------------------------------------
(10)
(11)    #ifndef __CinterAgentConnector_idl__
(12)    #define __CinterAgentConnector_idl__
(13)
(14)    #include "cagentroot.idl"
(15)    #include "cproactiverole.idl"
(16)    #include "creactiverole.idl"
(17)    #include "ccouplingprotocol.idl"
(18)    #include "ciacstate.idl"
(19)    #include "cagentsystem.idl"
(20)    #include "cagent.idl"
(21)    #include "ciacfilter.idl"
(22)
(23)    // ----------------------------------------
(24)    // Klasse CinterAgentConnector
(25)
(26)    interface CinterAgentConnector: CAgentRoot {
(27)
(28)            attribute CReactiveRole Destination;
(29)            attribute CProactiveRole Entry;
(30)            attribute string Description;
(31)            attribute enum Type;
(32)            attribute enum Mode;
(33)            attribute int MaxConn;
(34)            attribute int MinConn;
(35)            attribute CCouplingProtocol Interaction;
(36)
(37)            attribute string Result;
(38)
(39)            attribute CIAState Status;
(40)
(41)            attribute CAgentSystem Affiliation;
(42)            attribute CIACFilter Filter;
(43)
(44)            void SynchronizeInterconnectionPoints
(45)                 (in CCouplingProtocol interaction);
(46)            int GetNumberOfConnections();
(47)            CIACFilter AssignFilter
(48)                      (in CCouplingProtocol interaction);
(49)            int ComputeMaxConnections()
(50)            CIACState ReportState()
(51)            CAgentSystem GetAffiliation(in CAgent agent)
(52)            void RegisterConnectiontype(in enum mode);
(53)            int AbortConnection();
(54)            void ReleaseConnection();
(55)            void ResetConnection(in CProactiveRole entry,
(56)                 in CReactiveRole destination,
(57)                 in enum mode);
                void TimeOut();
        };

        #endif    // CInterAgentConnector_idl
```

Code 1: IDL specification of an interagent connector

4.2 Compiling the view pattern

Table no. 2 compiles the information of the engineering examinations on interagent connectors into an architectural view pattern. The pattern represents complex knowledge and is reusable for deriving special classes of interagent connectors.

The special classes of interagent connectors `CIAInvocationConnector`, `CIAResourceConnector`, `CIATaskConnector`, `CIAInterfaceConnector` can be understood interpreting the view pattern.

5 Conclusion

For the engineering substantiation of partial systems in architectural frames architectural views are described and derived via a methodical process resulting in architectural view patterns. Architectural view pattern comprise of complex knowledge of architects and serve the enlightening of detailed information on properties, specifics, application techniques and semantics of building elements. The approach of architectural view patterns results in a specification and thus suits the overhanding of the results of engineering analysis and classification to software developers.

For classes of software application systems common architectural views can be prefabricated.

The complex approach of architectural view patterns was applied for interagent connectors. The architectural view pattern on interconnections in multi-agent systems comprises of detailed information on interagent connectors represented again by complex models and patterns [13].

Table 2. Architectural view pattern for interconnections in mas

Pattern: PInterconnctionView	
Class of software application system:	Multi-agent systems.
Partial system:	Service integration.
Building element:	Interagent connector.
Stakeholders:	Middleware specialists. Analysis, modeling, implementation of communication resources.
Application techniques:	Traversing the phases of construction, local and global matching, tuning of assignment, interaction of corresponding parties and destruction.
Application semantics:	Dynamic behavioral assignment of providing ports to requesting ports.
Properties and specifics:	Composite class model (figure no. 7).
Platform services	Special interagent connection service.
Program technical realization:	Template available via code no. 1.

6 Outlook

The concepts of the architecture-based approach as well as the principles, building elements and patterns will be applied, validated and extended within in a German fundraised project dealing with information logistics in hospitals "Agent system architectures for active medical documents – information logistics in multi-contextual domains". For further details on the architecture type-based approach as well as on ADE please refer to the URL <`http://www.cs.uni-potsdam.de/soft/taxt/`>.

References

[1] Allen, R., Garlan, G. 1997. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, Vol 6, No 3, 213-249.

[2] Buschmann, F., Meunier, R., Rohnert, H., Sonnerlad, P., Stul, M. 1996. *Pattern-Oriented Software Architecture. A System of Pattern.* Chichester: John Wiley & Sons.

[B] Hilliard, R. 2000. Viewpoint Modeling. 1st IVSE Workshop on Describing Software Architecture with UML.

[C] Hofmeister, C., Nord, R., Soni, D. 1999. *Applied Software Architecture.* Addison-Wesley.

[5] Horn, E., Schubert, W. 1993. *Objektorientierte Software-Konstruktion*. Hanser.

[6] Horn, E., Kupries, M. 1998. The use of object-oriented software architecture types for the substantiation, development and pre-fabrication of software agent systems, Position Paper, Special Interest Group on Methodologies and Software Engineering for Agent Systems, Amsterdam, The Netherlands.

[7] Horn, E., Kupries, M., Reinke, T. 1999. Properties and Models of Software Agents and Prefabrication for Agent Application Systems. In Proceedings of the Thirty Second Annual Hawaii International Conference on System Sciences, Wailea, Hawaii.

[8] Horn, E. 1999. Agentenmodelle der Softwaretechnik. In Tagungsband des Workshops Multiagentensysteme, Tutzingen, Deutschland.

[9] Horn, E., Reinke, T. 2000. Musterarchitekturen und Entwicklungsmethoden für Multiagentensysteme. *Künstliche Intelligenz*, Vol 4.

[A] IEEE. Std 1471. 2000. Recommended Practice for Architectural Description of Software-Intensive Systems.

[D] Kruchten, P.B. 1995. The 4+1 view model of architecture. IEEE Software, Vol.28, No.11.

[8] Kupries, M. 1999. ADE: An Architecture type-based Development Environment for Agent Application Systems. *AgentLink*, No 2, 12-14.

[9] Kupries, M. 2000. Interagent Connections in Multi-Agent Systems. Dissertation. University of Potsdam.

[10] Nwana, H.S. 1996. Software Agents: An Overview. Knowledge Engineering Review, Vol.11, No.3, 205-244, Cambridge University Press.

[11] Shaw, M., Garlan, D. 1996. *Software Architecture. Perspectives on an Emerging Discipline.* Upper Saddle River: Prentice Hall.