View composition in multi-agent architectures

Nelis Boucké*, Tom Holvoet

DistriNet lab, Department of Computer Science,
Katholieke Universiteit Leuven (KULeuven), Belgium
E-mail: {nelis.boucke,tom.holvoet}@cs.kuleuven.be

*Corresponding author

Abstract: Our experience in research and development on a multi-agent architecture for an industrial Automatic Guided Vehicle Transportation System (AGVTS) reveals a crucial problem w.r.t. the specification of software architectures. In particular, there is a lack of support for relating and composing multiple views in architectural descriptions. This lack hampers changeability, as it prevents separating a concern while maintaining a good overview of the architecture as a whole.

In this paper, we argue for explicit support in relating and composing architectural views. We propose three types of relations that allow specifying and composing structural views. To validate and evaluate our work, we extend a general-purpose architectural description language, xADL. We assess the usability and consequences of structural view relations in the context of the multi-agent architecture for the AGVTS. We learn that view relations and composition improve changeability of a multi-agent architecture by offering explicit support for (1) separating concerns in separate views and (2) composing the overall architecture from these separate views. The tradeoff for this improved changeability is the additional cost of documenting relations between architectural views. Nevertheless, documenting view relations give rise to architectural descriptions that are more expressive and less ambiguous.

Keywords: composition, relations, multi-agent systems, multi-agent architectures, architectural description language (ADL), separation of concerns.

Reference to this paper should be made as follows: Boucké N. and Holvoet T. (2007) ‘View composition in multi-agent architectures’, Special issue on Multi-agent systems and software architecture, International Journal of Agent-Oriented Software Engineering (IJAOSE).

Biographical Notes: Nelis Boucke is a PhD student at DistriNet Labs since 2004. His research interests are aspect-oriented software development, multi-agent systems, and software architecture. The focus of his PhD research is on investigating ADLs to improve separation of crosscutting concerns in multi-agent architectural design.

Tom Holvoet is a full time professor in the Computer Science Department at the Katholieke Universiteit of Leuven, Belgium. He obtained his PhD in 1997 for work in open concurrent software development. Since then he is active in research on decentralized systems, aspect-oriented software development, and middleware. Tom leads a research group with seven PhD students within the DistriNet Lab.

1 INTRODUCTION

The architecture of a software system defines the design structure or structures, which comprise software elements, the externally visible properties of those elements, and the relationships between them [4]. Modeling a software architecture is an essential step for the development of complex systems, including multi-agent systems. During the last decades, several Architectural Description Languages (ADLs) have been proposed for modeling architectures, such as ACME [27], xADL [19], AADL [48], Clements et al. [15] and Rozanski et al. [45]. An architectural description typically consists of several views. A view defines a set of software elements and the associated relationships between these software elements [15]. Views are used for representing the system or a part of the system from the perspective of a (related set of) concern(s) [30]. The core issue in architectural design is to define the views that are relevant for key stakeholders' concerns.

The problem central to this paper is that there is a lack of support in ADLs to compose and relate views with each other. Once parts of the architecture are separated using the views, there is a need to compose the views to obtain the architecture as a whole, and this is where the current architectural practice falls short. Composition starts with identifying the *relations* between views. We use the term *composition* for an aggregation of several views and relations between views. As a consequence, the *architecture* of a system is a composition of all views and associated relations between the views.

Experience shows that such support is essential to build a multi-agent architecture for a realistic industrial application. Concretely, we encountered the need for composition during an attempt to describe distribution in a separate view to improve changeability and at the same time keep a good overview of the architecture as a whole, and this in the context of a multi-agent architecture for an industrial Automatic Guided Vehicle Transportation System (AGVTS).

Current architectural practice [15, 19, 27, 30] provides good support for describing several views, but the support for describing relations between views is too limited. We believe that composing and relating the views is as important as specifying the views. Consequently, we advocate that relations and compositions should become first class, and become an explicit and integral part of the architectural description language.

We put forward view composition and three concrete relations, i.e. *refinement*, *mapping* and *unification*, and use these relations for composing several structural views. As an illustration, we extend the language definition of a representative architectural description language (ADL) with explicit support for composing structural views. In this paper, we limit ourselves to adding explicit compositions between one type of view only, namely the structural view of the existing xADL [18]. xADL is a general purpose, extensible ADL for modeling architecturally-relevant structures. xADL supports the specification of structural views, but there is little to no support for the specification of relations between views. The xADL language is used as an example, but similar composition rules could be of use to other ADLs as well.

The contribution of this paper is threefold. Firstly, we identify and illustrate crucial shortcomings in current practice of architectural description in a multi-agent architecture. These shortcomings relate to the description of compositions and relations between views. Secondly, we put forward three concrete relations that can be used in compositions to bridge this gap. We extend the language definition of xADL to illustrate the concepts. Thirdly, we apply and evaluate the extended language in a realistic and concrete multi-

agent case study to enable describing distribution in a separate view and facilitate composing views to obtain a good overview of the architecture as a whole.

The paper is structured as follows. Section 2 illustrates the need for compositions and relations in the context of a concrete case study. Section 3 introduces view composition and relations, and illustrates the concepts by extending the xADL language definition. In section 4 we apply the extended xADL language to model an excerpt of the AGVTS and we evaluate view composition in this context. Section 5 discusses the results and the lessons learned. Section 6 discusses related work. Finally, section 7 presents the conclusion.

2 PROBLEM ILLUSTRATION IN A MULTI-AGENT ARCHITECTURE

In this section we illustrate the problem outlined in the introduction in the context of a concrete industrial case study build with an multi-agent architecture. Section 2.1 introduces the case study and its architecture. Section 2.2 discusses our experiences with this architecture and sketches the context in which we encountered the need for compositions and relations between views. Section 2.3 concretely illustrates the lack of support for composition and relations between views.

2.1 A Multi-agent Architecture for an Automatic Guided Vehicle Transport System

An Automatic Guided Vehicle Transport System (AGVTS) is a fully automated industrial system that uses multiple AGVs to transport loads in a warehouse or production plant. An AGV is an unmanned transportation vehicle. The AGVTS uses an external software system –called Ensor– that runs on the computer of each AGV vehicle to steer the vehicle, based on high-level commands (like move, turn) issued by the AGVTS. The main functional requirements for the AGVTS are: (1) allocating transportation tasks to individual AGVs; (2) performing those tasks; (3) preventing conflicts between AGVs, e.g. on cross-roads; and (4) charging the batteries of AGVs before they are drained. Transportation tasks are issued by a Warehouse Management System (WMS).

The AGVTS is build with a multi-agent architecture, in a joint research project between the research group Distrinet and Egemin [22], a producer of automated logistic systems [54]. The architecture decentralizes control and answers to new market demands, asking for flexible, open and robust systems that can handle dynamic operating conditions in an autonomous way. These qualities are ascribed to multi-agent architectures [8].

Using a multi-agent architecture means structuring the system into several autonomous agents connected to an environment. We follow the reference architecture for situated multi-agent architectures [51]. An agent component operates autonomously, but coordinates with other agent components to achieve the system goals. Each agent has three required interfaces: (1) a perception interface to perceive the environment; (2) an action interface to influence the environment; (3) a communication interface to communicate with other agents by sending and receiving messages through the environment. The environment is a software component, shielding the agents from details of the real world and providing the interfaces for perception, action and communication to the agents [53].

In the AGVTS we identified three types of agents. Firstly, an AGVAgent for each AGV vehicle, responsible for managing the tasks. Secondly, a TransportAgent for each transport in the system, responsible for searching an appropriate AGV to perform the transport and to follow up the dispatching. Finally, one or more TransportManagerAgents, in charge of

Figure 1 Three structural views of the original architectural documentation, extracted from [6]. Left: Decomposition in two subsystems. Middle: Internal structure of the AGVControl system. Right: Internal structure of TransportBase.

communicating with a WMS and managing (create, remove, etc.) the TransportAgents. The tasks of the environment include the translation of high-level commands from the AGVAgent to Ensor (the software package for low-level control of the AGV vehicle).

2.2 Experiences

In this section we discuss the experiences with a multi-agent architecture for the AGVTS that motivates describing distribution in a separate view.

The initial architecture [6] decentralizes control by using several agents and physically distributes this control by locating the agents on different computer systems. We refer to this as decentralized deployment. After building a proof-of-concept demo^a, Egemin realized that using this architecture is a big step with far reaching effects for the company. To limit risks, the partners agreed on a step-wise distribution. Initially, all agents are located on a single computer system with remote access to the AGV vehicles. We refer to this as centralized deployment. Later on, the agents can be step-wise distributed. Such schema enables to deploy and test multi-agent architectures without being bugged with typical problems of a physically distributed system. We can conclude that distribution will need to change several times.

A desirable property of the architecture is that the impact of changing distribution is as small as possible, but this is not the case. Analysis reveals that the decisions with respect to distributions are scattered over the views, both over multiple views of the same type (i.e. several structural views) and over multiple views of different types (i.e. both structural, behavioral and deployment views). Additionally, the decisions are tangled with decisions for other concerns. We briefly illustrate scattering and tangling between distribution and control using two structural views shown in fig.1. Control covers the decisions to decompose the system in agents and an environment, leading to a decentralized architecture in which each agent controls its own behavior. A structural view shows the types of components and connectors and emphasizes on how they should be linked together. The AGVTS view defines two subsystems that are needed for distribution. The AGVControl view and the TransportBase view describe the internal control structure of the respective subsystems. Thus both views are influenced by distribution and control, effectively mixing up the

^aSome videos on the proof-of-concept application running on real AGV vehicles can be found on [23].

Figure 2 Left: Control view, showing the decomposition into agents and environment. Right: Distribution view, showing the decomposition into subsystems for deployment. Same graphical notation as fig. 1.

two concerns. Similar scattering and tangling appears in other views of the architectural description.

As a consequence, the architects judged that separating distribution from the remainder of the application was necessary, leading to the development of a second version of the architecture. It was during the development of this architecture we encountered the problem that is the subject of this paper, discussed next.

2.3 Problem: Lack of Support for Appropriate Relations Between Views

During the development of a second version of the architecture that described distribution in a separate view, we encountered the lack of support for composition and relations between views. A long known engineering principle states that having divided to conquer, we must reunite to rule [32]. Once distribution, as a concern, is separated using the views, there is a need to compose these views together to obtain the architecture as a whole. We claim that this is where the current architectural practice falls short.

We briefly illustrate the need for relations using two concrete views in fig. 2. The figure shows two structural views, the Control view and the Distribution view^b. The left part of the figure shows the Control view, outlining a decentralized control architecture defining the types of agents and an environment. The right part shows the Distribution view, defining two subsystems (AGVControl and TransportBase), to be deployed on an AGV vehicle and a server respectively.

Imagine an architectural description containing these two views, without any relations. Much remains unclear, implicit and ambiguous. We illustrate this with several examples. Firstly, it is unclear how the subsystems of the Distribution view are related to the agent and environment of the Control view. The two views both contain the Ensor and WMS components, which could lead to an interpretation that the components in the views are linked together through these components. This is not what the architect wanted to communicate. Secondly, someone might know or guess that the subsystems contain parts of the Control view, and that the AGVAgent is located on the AGVControl system and that the other agents are located on the TransportBase. But even with this information it

^bThese two views describe a similar architecture as the views in fig. 1, but the description is structured to separate control and distribution.

Figure 3 Illustration of overlap between both views. The dashed circles point out the overlaps between the elements. The semantic meaning of such overlap is unclear. For the remainder we use the same graphical notation as fig. 1..

remains unclear what this means for the environment component or how to interpret the connector between the *TransportManagerAgent* and the *WMS*.

At first sight, an obvious solution is to describe the components of both control and distribution in a single view. But this is not an acceptable solution in this case, since it prevents separation of distribution in the architecture.

Towards a solution. Relating these two views is not easy. To illustrate the challenges, we constructed a view containing an intuitive mix of components of both Control and Distribution in fig. 3. We identified three specific challenges from which we know from experience that they are quite common in architectural descriptions.

1. Express that elements that appear in different views are actually the same element. Examples are the *Ensor* and the *WMS* components.
2. Express that a component of one view is a subcomponent of a component of another view. For example the *AGVAgent* is a subcomponent of *AGVControl* component.
3. Express that there is a partial overlap between components, and solve this overlap. For example, the *Environment* component overlaps the two subsystems *AGVControl* and *TransportBase* of the Distribution view. Figure 3 illustrates this.

Important for each of the challenges is that the relations are described explicitly and unambiguously. For example, describing that the *TransportManagerAgent* is a subcomponent of the *TransportBase* is not enough, since it leaves open how the connector between the *TransportManagerAgent* and *WMS* relates to the connector between the *TransportBase* and the *WMS*. In this case, a more exact specification of the interplay between sub- and superelements is needed, such as which port of a subelement maps on the port of an the super element. Without such unambiguous and fully specified relations there may be unexpected effects of changing one view to another view, leading to inconsistencies in the architectural description.

Architectural practice. Currently, there are only few approaches in architectural practice that support some kind of relations between views. As far as we know, none of these support explicit and direct relations between views to cope with the three challenges outlined above. Moreover, the conceptual model of the IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [30] establishes terms and concepts pertaining to the content and use of architectural descriptions, but lacks the concepts of relations between or compositions of views. The document only briefly mentions that any inconsistencies between views should be documented, but leaves completely open how this should be done.

ADLs like ACME [27] and xADL [19] allow describing multiple views of the same type, but lack support to directly relate structural views with each other.

Clements et al. [15] uses tables to define a mapping between elements in different views defining a simple type of relation. Yet, this well known book on architectural documentation only briefly discusses relations. Drawbacks of these types of relations are that they remain informal (relying on a textual explanation) which may lead to ambiguities, and the support for different types of relations is limited. The first two challenges can be partially expressed in such tables, by taking down which elements are the same and which elements are subelements of other elements. But even then the description is ambiguous since much is left open on the exact interplay between the elements. Expressing the third challenge in a useful way is even harder.

Rozanski et al. [45] only mentions that different types of views might be interrelated with each other. The conceptual model [45, pag. 33] allows describing several views of the same type, but the work provides no support for describing relations between views. The authors only offer an intuitive checklist between each pair of types of views that an architect can follow to manage the consistency of the architecture.

3 VIEW COMPOSITION

This section adds view composition and relations as a first-class concept in an architectural description, and by this provides support for explicit relations between structural views.

Concretely, in section 3.1 we present three types of relations based on the challenges identified in the previous section. Section 3.2 presents the xADL language definition and extends this definition with support for composition and these three relations. Section 4 applies the extended xADL language to redesign a part of the AGVTS architecture. View composition is partially inspired by the work on aspect orientation, discussed in section 6.

3.1 Compositions of Structural Views

A composition is an aggregation of relations. To cope with the challenges identified in section 2.3, we introduce three types of relations.

- **Unification:** Unifies elements (components or connectors) from different structural views with each other, i.e. expresses that elements that appear in the different views are actually the same element. The unified elements must either have exactly the

same interfaces, or the architect must define the corresponding interfaces. Unification is used for the first challenge.

- **Mapping:** Maps individual elements or groups of elements (called subjects) from one structural view on a single element of another structural view (called target). The subjects then become subelements of the target element. The points on which the elements are connected to each other are the interfaces (we call this joinpoints). An architect must define corresponding interfaces between the target and the subject. Mapping is used for the second challenge.
- **Refinement:** Defines that a specific structural view (referred to as inner structure) describes a substructure for an architectural component (referred to as outer component) of another structural view. The joinpoints used in the refinement relation are the interfaces, the architect must define corresponding interfaces between the outer component and interfaces in the inner structure. Refinement is used for the second challenge.

The third challenge is achieved by a combination of the relations. We come back to this while illustrating the relations in section 4.

3.2 Adding View Composition to xADL

We will use the xADL language in our illustrations. Motivations to use the xADL language are that it provides a core language with a simple set of general purpose constructs

Figure 4 Example xADL specification, showing the Agent and Environment components, connected with the ActionCon connector. We used the same abbreviated notation as the xADL authors used in [19] to improve readability of the specification.

to model an architecture. Additionally, it provides an easy way to extend the language definition and the associated tool support for new concepts, allowing fast prototyping of new ADL constructs.

xADL supports three types of views: (1) a view defining component types; (2) structural views describing component and connector types and how they are linked to each other; (3) instance views describing component and connector instances and how they are linked to each other. Both the structural and instance view are a kind of component and connector viewtype in the sense of [15]. But the structural view shows types of components and connectors emphasizing on how they should be linked together (called structure by the xADL authors). The instance view uses component and connector instances to show the dynamics of the system (called instance). In this paper, we focus on the second type of view of xADL and call this, in line with the terminology suggested by the xADL authors, a structural view on the system.

In section 3.2.1 we introduce the basic building blocks of xADL and show how they are represented in the architecture. Section 3.2.2 discusses the language definition of xADL. Finally, section 3.2.3 extends the language definition with compositions and relations.

3.2.1 Basic Elements of xADL

An structural view in xADL is built around the following basic elements^c.

- **Components:** Components are the loci of computation in the architecture. Components have a unique identifier and a textual description, along with a set of interfaces.
- **Connectors:** Connectors are the loci of communication in the architecture design. Similar to components, connectors also have a unique identifier, a textual description, and a set of interfaces.
- **Interfaces:** Interfaces are components' and connectors' portals to the outside world. A "interface" in xADL is called "port" for components and "roles" for connectors in other languages like ACME ADL. Interfaces have a unique identifier, a textual description, and a direction. The direction indicates whether the interface is provided, required, or both.
- **Links:** Links are connections between interfaces, defining the topology of the architecture.

Figure 4 shows an example architectural description containing two components (Agent and Environment^d) one connector, ActionCon, and links to attach the connector to the components. In principle, the identifier of an interface can be any unique string. We follow the convention that the identifier uses the name of the element dot interface name to make the description more readable, like Agent.action. References are represented by the href property, containing a unique identifier or an element after the sharp symbol.

xADL is supported by a tool set integrated in the Eclipse platform, called ArchStudio^e. ArchStudio contains both a visual editor, that allows to graphically manipulate the architecture (notation similar to fig. 4(b)) and a special-purpose specification editor, representing the architecture in the form of a tree.

^cBased on [19]. The group element is left out because it is not used in this paper.

^dThe environment in this picture is a software component providing operations to the agents to view and manipulate the world.

^eWe used ArchStudio 4.0, available since November 2006 [31]

Figure 5 Overview of the XML schema defining the structural view of xADL.

3.2.2 Language definition of xADL

The xADL language is defined in the XML Schemas Definition Language (XSD [50]). XSD is a language to describe the syntax and structure of an XML-document. The xADL language definition consists of about ten different XML schema files, we will only focus on the part of the language defining *structural* views.

Concretely, we will make use of two XSD files defined by the authors of xADL. The `instance.xsd` file ('archinstance' namespace) is historically the first file of the xADL description and contains the definition of several common elements. Next to this, it contains the definition of run-time views. The `type.xsd` file, with as namespace 'structures', adds both types and structural views (the `archStructure` element and `ArchStructure` type) to xADL.

Figure 5 shows a visual representation of an extract of the XML schema defining the type of elements in a structural view. The `ArchStructure` type defines what type of elements an `archStructure` element can contain. The example in fig. 4 shows that such structure is a subelement of the `xArch` element, but this is not shown in our overview. As an example of how the XSD file looks, we included a small excerpt description of the `archStructure` element and `ArchStructure` type:

```
<xsd:element name="archStructure" type="ArchStructure"
minOccurs="0" maxOccurs="unbounded"/>

<xsd:complexType name="ArchStructure">
  <xsd:sequence>
    <xsd:element name="description" type="archinstance:Description"/>
    <xsd:element name="component" type="Component" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="connector" type="Connector" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="link" type="Link" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="archinstance:Identifier"/>
</xsd:complexType>
```

The arrow between the sequence element of `ArchStructure` and `Description` means that an element `archStructure` (of type `ArchStructure`) starts with an element `description` (of type `Description`). We call `description` a subelement of `archStructure`. If no multiplicity

Figure 6 Overview of the XML schema definition of architectural composition, our extension to the xADL language. The full definition file can be found in the composition.xsd file on <http://www.cs.kuleuven.be/~nelis/xAdlExtension/>. Same graphical notation as fig. 5.

is given, the element occurs exactly ones. Otherwise the arrow is annotated with minimal and maximal occurrence, in which star stands for unbounded. The arrow can be annotated with the name of the element. If no name is given, the element has the name of the type, but starting with a lowercase letter (e.g. the element is called description if its type is Description).

In short, an ArchStructure type consists of a sequence of an element description (type Description), zero or more component elements (type Component), zero or more connector elements (type Connector) and zero or more link elements (type Link). Finally, it has an attribute containing a unique identifier. The description element contains a textual description of the structure.

3.2.3 View composition into the xADL language

To make the composition a first-class element, we add a subelement to the xArch element, called archComposition. The internal structure of this element is defined by the ArchComposition type shown in fig. 6. The arrows leading to the Description type are omitted from the figure to make it more readable, the types ArchComposition, Refinement, Unification and Mapping all have exactly one description element.

Figure 7 shows an example of an architectural composition using the refinement composition operator. The figure shows three structures (represented by a UML package symbol: Struct1, Struct2 and Struct3) and a composition specification. The composition specifies that Struct2 refines component A of Struct1, resulting in Struct3. After the mapping, the elements of Struct2 are subelements (subcomponents or subconnectors) of component A. The joinpoints between the structures are the interfaces, specified in the interface mappings part of the refinement composition operator. For example, the A.up interface of component A is mapped on interface Z.up of its subcomponent Z. Currently, the composition between different structures is done by hand. Because the composition specification is explicitly specified in XML, appropriate tool support could make applying a composition really easy. This needs further investigation.

Figure 8 shows an example of a composition specification in which the mapping and unification relations are used to relate three structural views. The composition specifies that several elements of Struct5 are mapped on (are subelements of) component A and

Figure 7 Example composition using the refinement composition operator. Struct2 refines component A of Struct1, resulting in Struct3. Same graphical notation as fig. 4.

Figure 8 Example composition using the mapping and unification relation. Parts of Struct5 are mapped on component A and B of Struct4, resulting in Struct6. Same graphical notation as fig. 4.

Structural view	Aim	Step
Control	Describes the agent types, environment and associated relations.	1
Centralized	Subsystems needed for centralized deployment.	2C
EnsorConnectorInternal	Describes a refinement of the EnsorConnector.	3C
ControlExtendedAlt1	Composition of Control and EnsorConnectorInternal.	3C
CentralizedAGVSystem	Composition of ControlExtendedAlt1 and Centralized.	4C
Decentralized	Subsystems needed for decentralized deployment.	2D
EnvironmentInternal	Describes a substructure of the Environment component.	3D
ControlExtendedAlt2	Composition of Control and EnvironmentInternal.	3D
DecentralizedAGVSystem	Composition of ControlExtendedAlt2 and Decentralized.	4D

Figure 9 High-level overview of the composition between the views.

component B of Struct4, resulting in Struct6. The joinpoints between the structures are the interfaces, specified in the interface mapping part of the refinement composition operator. For example, the *B.up* interface of component B is mapped on interface *X.up* of its subcomponent X.

Notice that interfaces are the level of abstraction on which the views are bound to each other. The relations rely on specifying correspondence between interfaces (interface mapping). This restriction to one-to-one mapping might not work well in some cases. For example, one structural view could contain two interfaces which map on one interface in another structural view. In some situations there will be no other choice but refactoring the interfaces of one of the structural views.

4 APPLYING VIEW COMPOSITION IN PRACTICE

In this section the extended xADL language is applied to an excerpt of an architectural description for the AGVTS. The new architectural description for the multi-agent architecture is built up step-by-step in section 4.1. It separates an essential concerns of a multi-agent architecture, namely distribution, from the remainder of the system. We model both centralized and decentralized deployment, leading to two different systems. Section 4.2 evaluates this architecture and analyses the impact of changing distribution, comparing the new architectural description with a previous version.

4.1 Example of Composing Structural Views in the AGVTS

We start with a high-level overview of the process and subsequently describe each step in detail.

Figure 9 provides an overview of the structural views of the new architectural description. The left hand side shows a table with a short description of the aim and the step in which the structural view is described. The right hand side provides a high-level overview of the composition between these views.

We explain the process of building a multi-agent architecture for the AGVTS using this overview.

Figure 10 The Control view showing the agent types, the Environment component and the relations with external components (WMS and Ensor). To simplify the figure, we only show the names of connectors and interfaces used in the composition specifications.

- First, we defined two separate views to cope with respectively control and distribution, going out from centralized deployment. These two views are developed in isolation, called Control view in step 1 and Centralized view in step 2C respectively (the C in 2C stands for Centralized).
- Next we want to define relations between the two views. Here we see that the structures of the views do not fit easily, because there is a partial overlap between the EnsorConnector and the subsystems. This relates to the third challenge of section 2.3. The solution is to break up one of the components to make it fit. Therefore, we introduce an auxiliary view that breaks up the EnsorConnector called the EnsorConnectorInternal view in step 3. We compose this view with the Control view leading to ControlExtendedAlt1, which is now better suited for relations with the Centralized view.
- Finally, Step 4C relates the result of 3C with the Centralized view, leading to a centralized multi-agent architecture for the AGVTS (CentralizedAGVSystem).

Steps 2D till 4D describe a similar process, but for the distributed deployment schema, leading to a decentralized multi-agent architecture for the AGVTS. The D after 2D till 4D stands for Decentralized. Next, we describe each of the steps in detail.

4.1.1 Step 1 - Control

The Control view shown in fig. 10 has already been explained in detail in section 2.3 and is the same as the left hand side of fig. 2. The view shows a decomposition in agents and an environment.

4.1.2 Step 2C - Centralized

The goal of the first distribution schema (centralized) is to allocate all agents and the environment to a single server. To allow such allocation, we must provide remote access to the Ensor component which is situated on the AGV vehicle.

Figure 11 The Centralized view, showing the subsystems used for centralized deployment.

Figure 12 Left: EnsorConnectorInternal view. Right: Composition specification to compose the EnsorConnectorInternal (at the left) and the Control view (fig. 10).

We start with defining two subsystems, shown in fig. 11, Controller and RemoteEnsor. They will be deployed on a single server and an AGV vehicle respectively. Both have an interface for remote communication (connected to each other). The Controller is connected to the WMS, RemoteEnsor to the Ensor system.

Next, we want to define the relation between Centralized view and the Control view. Yet, there is partial overlap between the EnsorConnector and the Controller and the RemoteEnsor components. The connector between Environment and Ensor does not have appropriate support for remote communication yet. This relates to the third challenge of section 2.3. To cope with this, we refine the EnsorCon in the next step.

4.1.3 Step 3C - EnsorConnectorInternal

To refine the EnsorConnector, we first define an auxiliary view and then compose this auxiliary view with the Control view.

Figure 12 shows an auxiliary view called EnsorConnectorInternal defining the internal structure of the EnsorCon. This structural view contains two subcomponents connected with a connector, EnsorProxy and RemoteEnsor respectively. EnsorProxy will act as a proxy for Ensor from the perspective of the Environment component.

Next, we compose this auxiliary view (EnsorConnectorInternal) with the Control view using the *refinement* relation. The specification of composition is shown at the right hand side of fig. 12. The composition results in the ControlExtendedAlt1 view that is used in the next step to compose with the Centralized view^f.

^fNotice that there is no explicit connector between Environment and EnsorProxy, or between RemoteEnsor and Ensor. At first glance this might seem strange, but the links between the respective com-

Figure 13 Final composition in centralized agent system for AGVTS. Left: composition specification composing the Centralized view (fig. 11) and the ControlExtendedAlt1 view (fig. 12). Right: the resulting view after this composition, called CentralizedAGVSystem.

4.1.4 Step 4C - CentralizedAGVSystem

Finally, the Centralized view is composed with the ControlExtendedAlt1 view in fig. 13. We use both *mapping* and *unification* for this. The EnsorProxy component, all agents and the Environment component are mapped on the Controller. We unify the pairs of RemoteEnsor, Ensor and WMS components with each other. This results in the CentralizedAGVSystem view (shown at the right hand side of fig. 13).

Notice that we solved an instance of the third challenge. The combination of relations and an auxiliary view allowed to compose the Centralized view with the Control view. We included the EnsorConnector in fig. 13 as an illustration. It still cuts across the borders of the Controller component, but through the subcomponents and interfaces it is clear what this means.

This wraps up the description of the first alternative for distribution, leading to a centralized system. The next step starts from the same Control view, but will build up the system going out from a decentralized distribution schema (resulting in another, decentralized system).

4.1.5 Step 2D - Decentralized

The goal of the second distribution schema is to distribute the agents. In such a schema, the AGVAgent is allocated on the AGV vehicle, the TransportAgent and TransportManagerAgent on one or several servers.

ponents pass the border of EnsorCon. This is a result of describing a refinement of EnsorCon, meaning that the interfaces of the EnsorCon are delegated to internal components.

Figure 14 Decentralized view. This is the same view as the right hand side of fig.2, with resized elements to allow a compacter presentation.

Figure 15 (a) EnvironmentInternal structural view. (b) Specification to compose EnvironmentInternal with Control.

Again, we start with defining two subsystems in fig. 14, AGVControl and TransportBase, to deploy on the AGV vehicle and a server respectively. Both have an interface for remote communication (connected to each other). The TransportBase has a connection to the WMS, the AGVControl component to Ensor.

Next, we want to define the relation between the Decentralized view and the Control view. There is a partial overlap between the Environment component and the subsystems. Notice that this is the example used to explain the third challenge. We will solve this overlap in the next step.

4.1.6 Step 3D - EnvironmentInternal

To refine the Environment component, we first define an auxiliary view and then compose this auxiliary view with the Controller view.

The left hand side of fig. 15 shows the auxiliary view defining the internal structure of the environment. The environment is split up in two local environments, AGVLocalEnvironment and TransportLocalEnvironment. These two local environments synchronize state, take care of communication, and together they are a virtual representation of a single Environment component.

Defining the composition with this auxiliary view using the *refinement* relation is less trivial than the previous example. The Environment is split in two components, but three interfaces (perception, communication, action) are offered by both components, requiring a more complex interface mapping. In the composition specification we provide several alternative mappings and specify which connector uses what alternative. The right hand side of fig. 15 shows the composition specification. The composition results in the

```

<archComposition id="DecentralizedAgentSystem">
  <description>Decentralized AGV agent system</description>
  <mapping>
    <target>
      <archStructure href="#DecentralizedSubSystem"/>
      <element href="#AGVControl"/>
    </target>
    <subStructure>
      <archStructure href="#ACEExtended2"/>
      <element href="#AGVAgent"/>
      <element href="#AGVLocalEnvironment"/>
    </subStructure>
    <interfaceMapping>
      <outerInterface href="#AGVControl.ensor"/>
      <innerInterface href="#AGVLocalEnvironment.ensor"/>
    </interfaceMapping>
    <interfaceMapping>
      <outerInterface href="#AGVControl.network"/>
      <innerInterface href="#AGVLocalEnvironment.network"/>
    </interfaceMapping>
  </mapping>
  <mapping>
    <target>
      <archStructure href="#DecentralizedSubSystem"/>
      <element href="#TransportBase"/>
    </target>
    <subStructure>
      <archStructure href="#ACEExtended2"/>
      <elements selectionStatement="Transport"/>
    </subStructure>
    <interfaceMapping>
      <outerInterface href="#TransportBase.network"/>
      <innerInterface href="#TransportLocalEnvironment.network"/>
    </interfaceMapping>
    <interfaceMapping>
      <outerInterface href="#TransportBase.wms"/>
      <innerInterface href="#TransportManagerAgent.wms"/>
    </interfaceMapping>
  </mapping>
  <unification><!-- Ensor --></unification>
  <unification><!-- WMS --></unification>
</archComposition>

```

Figure 16 Composition specification to form the decentralized agent system. The result is shown in fig. 17.

Figure 17 Decentralized agent system for AGVTS, result after composition in fig. 16. We left out the Environment component to simplify the figure.

ControlExtendedAlt2 view that is used in the next step to compose with the Decentralized view.

4.1.7 Step 4D - DecentralizedAGVSystem

Finally, we can compose the system to form a decentralized version of the AGVTS. Figure 16 contains the composition specification using the mapping and unification relations. This results in the DecentralizedAGVSystem view in fig. 17. Notice that this resulting architecture is very similar to the architecture of the original architectural description in fig. 1. The main difference is the way in which the architecture is described and the process followed to get there.

This wraps up the description of the second alternative for distribution, leading to a decentralized system.

4.2 Evaluation

We evaluate the use of view composition for the AGVTS architecture in two ways. Firstly, we extend an existing Architectural Tradeoff Analysis Method (ATAM [16, 33]) to analyze the key architectural decisions in section 4.2.1. Secondly, we analyze the impact of changing distribution in section 4.2.2. In both evaluations two architectures are compared with each other: the original architectural description and the architectural description that uses view composition as presented in section 4.1 (referred to as new architectural description).

4.2.1 Analysis of Architectural Decisions with respect to New Quality Scenario

The ATAM is a well known architectural evaluation method based on quality scenarios and stakeholder involvement [20], to evaluate whether an architecture satisfies the required quality attributes. The ATAM itself is a two day workshop that exists of several steps. There are steps covering the presentation of business drivers and architecture, the analysis of architectural approaches, the generation of a quality attribute tree and the analysis of architectural decisions with respect to the quality scenarios.

We specifically chose to use the ATAM because a previous evaluation of the AGVTS architecture [8, 6] used the same method, evaluating the decentralized architecture with respect to flexibility and openness. It was during one of the post ATAM discussions that we experienced the need for step-wise integration of distribution as explained in section 2.2. In this paper we extend the previous evaluation by: (1) formulating a new quality scenario to capture the need for step-wise integration of distribution in the architecture; and (2) analyzing the architectural decisions with respect to this new quality attribute.

Figure 18 shows the new quality scenario and the analysis of architectural decisions. The new quality scenario states that it must be possible to change the distribution of control (e.g. for stepwise distribution of the control software) with reasonable effort, i.e. without redefining the remainder of system. We want to stress that this new quality scenario forms an unanticipated change for the original architectural description. So it is only logical that this architectural description is not really suited for an unanticipated quality scenario. The motivation to perform the ATAM for the original architectural description is to identify the architectural decisions that form a risk *with respect to the new quality scenario*. The motivation to perform the ATAM for the new architectural description is to assess whether the new architectural decisions prevent such risks, relying on the expertise of the assembled stakeholders. For this extension to the evaluation we involved several stakeholders from the original evaluation team.

Each of the architectural decisions is (non-exclusively) categorized to be a sensitivity point, a tradeoff point, a risk or a non-risk [16]. A sensitivity point is an architectural decision that is essential for the achievement of the quality scenario. A tradeoff point is an architectural decision that is essential for several quality scenarios and thus involves balancing between several qualities. A risk is an architectural decision that (might) form(s) a problem for the achievement of the quality scenario. A non-risk is the reverse of a risk, i.e. it is a good architectural decision.

During the analysis of the architectural decisions of the original architectural description, the stakeholders identified three risks (for AD1 till AD3) and one non-risk. AD2 is obviously influenced by AD1, but it not a necessary consequence. That is why we considered it a separate decision. AD3 is a decision that is not explicitly made, but implicitly

Scenario: It must be possible to change the distribution of control (e.g. for stepwise distribution of the control software) with reasonable effort, i.e. without redefining the remainder of the system.

Original architectural description				
Decisions	Sensitivity	Tradeoff	Risk	Nonrisk
AD1 Physically distributed agents			R1	
AD2 Decentralization and distribution			R2	
AD3 Distribution first			R3	
AD4 Subsystems				NR1

AD1 During one of the first discussions on the architecture, the project partners agreed to physically distribute the agents. Consequently, the architectural team considered this as a constraint. This architectural decision is obviously a risk with respect to the quality scenario (risk R1).

AD2 The decisions to both decentralizes control by using several agents and physically distribute this control by locating the agents on different computer systems where taken together. Decentralized control and distribution are strongly coupled in several architectural views. A consequence is that the coupling between both concerns is implicit in the description of the views, hampering changeability of distribution. This is denoted as risk R2.

AD3 The top-level decomposition is based on distribution of control, the remainder of the architecture either implicitly or explicitly relies on this. As an example, fig. 1 in section 2.3 shows the three views of the original architectural description. The AGVTS view shows the top-level decomposition in two subsystems for distribution. The two other views, the AGVControl and TransportBase view, explicitly rely on this decomposition as they describe the internal control of the respective subsystems. Changing distribution implies that all dependent views need to as well. In the example, both the AGVControl and the TransportBase view become obsolete and the architects need to define new views from scratch. This decision is denoted as risk R3.

AD4 We explicitly model the components that represent individually deployed subsystems and their connection in structural views. This is denoted as N1, because it provides clarity on which connections are local and which ones are remote and it simplifies the deployment view.

New architectural description				
Decisions	Sensitivity	Tradeoff	Risk	Nonrisk
AD5 Separating distribution	S1	T1		
AD6 EnsorConnectorInternal	S2	T2		
AD7 EnvironmentInternal	S3	T3		
AD8 Explicit composition		T4		NR2
AD9 Subsystems				NR3

AD5 The description of distribution is separated using views. The Centralized (fig. 11) and Decentralized (fig. 14) views contain the description of distributed subsystems for respectively the centralized and decentralized version. These views are completely independent of control, described in the Control view (fig. 10). This is denoted sensitivity point S1 because it is essential for achievement of the quality scenario. This decision is considered as tradeoff T1 because separating distribution may make it more difficult to get a good view on the system as a whole. T1 is partly countered by AD8.

AD6 Some elements of the Control view (fig. 10) need to be refined to suit the centralized deployment schema, represented by the subsystems defined in the Centralized view (fig. 11). The architects decided to describe this refinement in a separate view called EnsorConnectorInternal (fig. 12, describing the internal structure of the EnsorConnector) and not in the Control or in the Centralized view. This is denoted as S2 because this makes changing the refinement easy, which is essential for the changeability of distribution. This decision is considered as tradeoff T2 because this may make it more difficult to get a good view on the system as a whole. T2 is partly countered by AD8.

AD7 Similar to AD6, some elements of the Control view need to be refined to suit the decentralized deployment. Again this is described in a separate view, called EnvironmentInternal (fig. 15) and denoted as S3. This decision is considered as tradeoff T3 because this may make it more difficult to get a good view on the system as a whole. T3 is partly countered by AD8.

AD8 The composition of the Control and the EnsorConnectorInternal view leads to the ControlExtAlt1 view. Later the ControlExtAlt1 view is composed with the Centralized view, leading the CentralizedAGVSystem view. For the new architectural description there is an explicit composition specification between related views and each of the architectural compositions leads to a new architectural view. This allows understanding the compositions by instantiating the resulting view and building the architecture stepwise manner, by using the resulting view in other compositions. This is denoted as non-risk NR2. This architectural decision does also involve a tradeoff between explicit specification and simplicity (T4). Building explicit specifications, like the one in fig. 16 incurs an additional cost. However, it was considered essential in this case because the stakeholders wanted an explicit specification of the relations between views.

AD9 Similar to AD4.

Figure 18 Analysis of architectural decisions with respect to a specific quality scenario, both for the original and for the new architectural description. AD stands for architectural decision.

resides in the architecture. Since there is a clear risk with respect to the quality scenario, we included this decision in the analysis.

For the analysis of the new architectural description, the architectural decisions AD5 till AD7 separate distribution from the remainder of the application. Such separation is needed to prevent the risks of the original architectural description. But each of these decisions involves the tradeoff that the separation may make it more difficult to get a good view on the system as a whole. That is why AD8 is necessary, cause explicit relations and compositions allow getting a better view on the system as a whole. As a consequence there is tradeoff, namely that explicit specification asks for an additional effort of the architect. Yet, the stakeholders agreed that such effort is needed to allow a consistent architectural description.

As a conclusion, we identified several risks in the original architectural description with respect to the new quality scenario. These risks are prevented in the new architectural description. The tradeoff is the additional cost of documenting relations between architectural views. But this is considered necessary by the stakeholder to cope with the prescribed quality scenario.

4.2.2 Impact Analysis of the Distribution Concern

Complementary to the ATAM, in which the evaluation is a social process relying on the experience of the assembled group of stakeholders, we experimentally measured the impact of changing distribution.

The analysis measures how many views of the original architectural description need to change when distribution changes and compares the results with the measurements of the new architectural description. This analysis is necessary to check our assumption that the new architectural description will need fewer changes than the original architectural description. We perform this analysis at two levels: detailed analysis of an extract of the architecture and a discussion on implications for the complete architecture. First, we study the impact of changing distribution for the extract of the architecture used in this paper.

The extract from the original architectural description consists of three views, shown in fig. 1. The first view shows the decomposition into subsystems for distribution. The two following views show the internals of these subsystems. To alter the architectural description from a decentralized deployment to a centralized deployment, we need to alter all three views. It is to be expected that the AGVTS view needs to change since the subsystems change, but building the two other views from scratch is a not desirable consequence. The AGVControl and TransportBase view have to be defined from scratch since they describe the internals of the respective subsystems for the decentralized deployment, non-existing in centralized deployment. Despite that, the control structure remains the same, the views in which this structure is described can not be reused.

The same extract from the new distributed architecture consists of five views and is presented in section 4.1. Three of them are constructed by the architect (namely Control in fig. 10, Decentralized in fig. 14 and EnvironmentInternal in fig. 15), the other two can be generated from these. From the three input views, two depend on distribution (namely Decentralized and EnvironmentInternal). Note that the EnvironmentInternal view is an auxiliary view influenced by both distribution and control. To alter the architectural description to centralized deployment, only two views and the associated relations need to be replaced (by the Centralized in fig. 11 and EnsorConnectorInternal in fig. 12). Notice that nothing changes to the description of the control structure, they are separately described in the Control view. The previously generated views must be regenerated.

As expected, the architect must alter fewer views for the new architectural description than the old architectural description, namely two instead of three. Moreover, for the original architectural description the architect must reason and reconstruct the control structure to capture it in new views, which is unnecessary in the new architectural description. A drawback is that changing the relations between views in the new architectural description will require an additional effort.

Next, we discuss the impact of changing distribution beyond the excerpt presented in this paper. The architectural description in [6] consists of more than twenty views. Nine views are influenced by both control and distribution (including the ones shown in fig. 1), six views only by control and four views only by distribution. So in total, thirteen views must be reviewed and possibly changed if the distribution schema is changed. But the nine views influenced by both control and distribution include the three of the excerpt of the architecture, from which we have shown that we could reduce the influence from distribution from two to one. This makes that there are six more views where distribution can be separated. This is an indication that the techniques presented in this paper may have a bigger impact on the architecture. Further research is required on this point.

5 LESSONS LEARNED AND DISCUSSION

The combination of separating distribution and view composition provided an added value to the architectural description of a multi-agent architecture for the AGVTS. It became possible to change distribution without changing the remainder of the architectural description (i.e. with no influence on the agent/environment decomposition). The results gathered in the real world case study provide first insights on possible advantages of view composition in general, pointing towards improved changeability and understanding of the architecture as a whole.

Lessons learned

- Distribution is a complex concern in multi-agent architectures. The combination of views and view composition allows to separate a concern in a multi-agent approach on the architectural level.
- Integration of multi-agent approaches with established engineering practices like software architecture and separation of concerns helped to assess the added value of the approaches. It has a positive influence on the way industrial people look to multi-agent systems and we are convinced that it stimulates industrial adoption.
- View composition and relations are a natural extension to architectural descriptions. Adding the concepts to the xADL language was possible without touching the existing elements of the language.
- The tradeoff for this improved changeability is the additional cost of documenting relations between architectural views. Nevertheless, documenting view relations gives rise to architectural descriptions that are more expressive and less ambiguous.
- The approach in this paper allows separating a concern in an *architectural description*. How the separation of concerns in the architectural description translates to the implementation is a subject of active research [3].

Types of relations. We proposed three types of relations. Each of the relations serves a specific purpose as outlined at the end of section 3. This set of relations was sufficient to cope with the challenges identified in the section 2.3. But this does not imply that the set of relations is complete. Possible other relations include relations to solve interferences between identifiers of software elements, relations to solve non-corresponding interfaces, etc. An important remark is that the total set of relations should be kept rather small so that it is easily comprehensible for a software architect.

Other types of views. This paper focuses on structural views and composition of structural views. Other types of views are also possible and relations between these views need further investigation. We briefly discuss first insights into the matter, distinguishing between relations between the same type of views and relations between different types of views.

Firstly, we discuss relations between the same type of view. We expect that each type of view may need its own set of relations. That is because each type of view has its own set of basic elements and different semantics, like component and connector types as building blocks of a structural view or processes and communication channels as building blocks for a process view. Relations are described in terms of the basic building blocks and are only meaningful within the specific semantics of that particular view. This seems to confirm our expectations.

Secondly, we discuss relations between different types of views. We suspect that each pair of types of views may need its own set of relations, for the same reason as explained above. Several relations between different types of views are already supported in ADLs. For example, in an xADL instance view an architect can describe that this component is of a particular type described in a structural view.

Notice that for an architectural description, not every type of view needs direct relations with all other types of views. Some relations between views can be indirect, i.e. through another type of view. Describing meaningful relations in general in the context of heterogeneous representations is a challenging problem [40], because there exist a large quantity of types of views each with its own notation and semantics.

Auxiliary views. To relate the Control and the Decentralized view, we used an auxiliary view called EnvironmentInternal view in fig. 15. An obvious question is why this auxiliary view is needed and why this is not described in the relations itself.

Describing a direct relation between the Control and Decentralized view is possible, but there will be a partial overlap between the Environment component and the subsystems of Distribution (third challenge, illustrated in fig. 3). With the partial overlap it remains unclear what part of the Environment resides on which subsystem and this may lead to ambiguities. From this we conclude that the Environment component must be broken up in order to unambiguously specify which part is a subcomponent of which subsystem.

We use relations to describe a connection between *existing elements*. Therefore breaking up the Environment component and defining subcomponents is not a responsibility of a relation, but can better be described in a separate view, which we call an auxiliary view.

Feasibility. Tool support is essential to make the approach feasible. We already mentioned that specifying explicit compositions will have an additional cost, but this is essential for architectural descriptions which are more expressive and less ambiguous. However, applying a composition, i.e. generating a new view from a composition specification, is currently done by hand and very cumbersome. Tool support for view composition, integrated in a

development suite of an ADL, is essential to make the approach feasible for use in practice. Implementing a proof of concept tool is one of our important paths for future work.

To find out how scalable the approach is needs further exploratory studies. Currently, we only considered an excerpt from the architecture of the AGVTS. Further studies are needed to see what the impact is in case of a large increase in views. Again, tool support will be essential before the approach can be applied for significantly larger architectural excerpts.

6 RELATED WORK

This section is subdivided in related work on: (1) architecture and multi-agent systems; (2) ADLs; (3) aspect orientation; and (4) other closely related approaches.

Architecture and Multi-agent systems. Several researchers use the classical notion of software architecture (based on [4, 15, 28]) and architectural design to develop multi-agent systems (MAS). Shehory [46] considers MAS from the perspective of architectural styles, to reason about the qualities that are typically attributed to the MAS styles. PROSA [9] offers a reference architecture for coordination in manufacturing control. The DistriNet research group defined a reference architecture for situated MAS [52] and developed an industrial AGV transportation system using MAS [54]. The software architecture of this system is modeled with different architectural views and structured according to different view types (module, component, deployment).

This paper differs from previous approaches in the sense that it promotes explicit separation of a concern –distribution– in architectural views and explicit composition between the views.

ADLs. Support for multiple views is considered essential in the architectural community, but not all ADLs have equal support for multiple views. Some older ADLs are invented before support for multiple views was considered important [38]. Other ADLs support one type and one instance view, but provide no support for multiple views of the same type. Examples are Rapide [37] and Wright [1]. A more recent ADL called AADL [48] also only supports multiple views of different type, but not multiple views of the same type. It uses packages as a way to separate concerns inside the views. The relations between the packages are directly described in the packages itself, i.e. the description of a package can directly refer to elements in another package.

The relations with ADLs that do support multiple views but lack support for relations between views [15, 45, 19, 27], are described in section 2.3.

Aspect Orientation. In the field of aspect orientation there is a lot of research on separation of concerns and compositions.

There are several aspect-oriented ADLs. Typically, aspect-oriented (AO) ADLs concentrate on defining aspectual components and innovative aspectual composition operators to add or change *behavior* on component interfaces [5]. In another words, they tend to mimic AO composition mechanisms similar to AspectJ [21] but on the level of component and connector interfaces [36]. Some representative examples of this category of ADLs are AspectualACME [24], DAOP-ADL [43], and PRISMA [42]. The distinguishing factor of view composition is the focus on *view* decomposition and composition in an ADL, based

on experience with a multi-agent architecture in an industrial-strength case study. Composition of views has been neglected in most of the available literature [5, 17] focusing on discussing the interplay of aspects and ADLs. An additional difference between existing architectural relations (aspectual binding, but also regular connectors and links) and the relations presented here is that the former describe relations between components or connectors while the latter describe relations between several structural views.

Some aspect oriented techniques for requirements engineering (e.g. [11, 39, 47]), detailed design (e.g. Theme/UML [13, 14, 12]) and programming (e.g. HyperJ [49]) provide similar ways of composing structures. The techniques stem from the principles of multi-dimensional separation of concerns (MDSOC) and the first instantiation of this philosophy in Java as introduced by Tarr et al. [49]. View composition could also be considered as an instantiation of MDSOC on the architecture. A view corresponds to a hyperslice, a composition to a hypermodule and our relations to the composition rules.

Several approaches combine aspect orientation with multi-agent systems. Kendall et al. [34, 35] describe the use of aspect orientation while designing and implementing role models for multi-agent systems (more recently continued in [10]). Contrary to this paper, the authors focus on implementation. In [26], Garcia et al. observe that several agent concerns such as autonomy, learning, and mobility crosscut each other and the basic functionality of an agent. The authors propose an aspect-oriented approach extrapolating implementation mechanisms to the architecture. More recently, Garcia et al. [25] identified crosscutting concerns for agent systems described in the ANote modeling language. Our work differs in its focus on relating architectural views. Boucké et al. [7] did some experiments to extrapolate the approach to the architectural level. The main conclusion was that the basic idea behind Theme/UML proved very interesting, but the language itself is less suited for architectural design. We introduce a language that is based on the same foundation (MDSOC, discussed before) but is embedded in an architectural description language.

Other closely related approaches. Nuseibeh et al. [40] developed a organizational framework in which different views and their relations can be made explicit. The approach focuses on identifying inconsistencies between several heterogeneous representations (e.g. different types of views). The approach in this paper differs by its focus on composition and by reification of these compositions in new views.

Atkinson and Kühne [2] propose architectural stratification to combine the strengths of component-based frameworks and model-driven architectures. Stratification is about gradual refinement of architectural structures based on architectural concerns, introducing interaction refinement as a relations between different architectural views. Our refinement relations is inspired by this work.

There exists a wide variety of relations in modeling languages. As an illustration, we end this related work with a brief comparison of the relations in UML.

There is a difference in the level of abstraction between relations in UML and relations in view composition. Relations as inheritance, aggregation and association are for example specified on the level of software elements, like classes. The relations specified in view composition are specified between architectural views. A view describes several software elements and how they are related to each other, and therefore it resembles more closely UML packages and diagrams than the UML software elements like classes, components, connectors, objects, states, etc. We successively discuss packages and diagrams.

Packages also describe several software elements and how they are related to each other. The differences between packages and views is that packages (internally) contain

explicit references to other packages, while views do not. Views are described in a declaratively complete way. They describe the system from the perspective of a particular concern and anything used to describe that concern is defined in the view itself. Declarative completeness has as advantage that views do not depend on each other, which promotes changeability [41]. But it may lead to some duplication and possible inconsistencies between the views, making relations even more important.

On the relations between UML diagrams, several works point out that the relations should be defined explicitly, since the lack of accuracy in their definition can cause evolution conflicts, inconsistency amongst different models and wrong interpretation or discussions about implicit relations [29, 44]. This is similar to our observations for architectural views.

7 CONCLUSION

From experience in research and development in multi-agent architectures we learned that distribution of control for an industrial system is a huge step and a common obstacle for the acceptance of multi-agent architectures. Step-wise distribution helped to accept such innovative architecture in an AGVTS, but requires that distribution can be changed easily. The combination of views and view composition offers changeability of distribution by offering explicit support for (1) separating distribution; and (2) composing the overall architecture from these separate views.

We have put forward three concrete relations that can be used in compositions to bridge this gap. We extended the language definition of xADL to illustrate the concepts. We applied the extended language in a realistic and concrete case study to enable describing distribution in a separate view and facilitate composing views to obtain a good overview of the architecture as a whole.

The results gathered provide insights on possible advantages of view composition in general, pointing towards improved changeability and understanding of the architecture as a whole. The tradeoff for this improved changeability is the additional cost of documenting relations between architectural views. Nevertheless, documenting view relations gives rise to architectural descriptions that are more expressive and less ambiguous. Another lesson learned is that this paper allows to separate a concern in an *architectural description*. How a separation of concerns in the architectural description translates to the implementation is a subject of active research.

There are several interesting tracks for future research. Firstly, there are interesting concerns for multi-agent architectures, such as coordination. An interesting challenge is to apply the technique to separate such concern on the architectural level. Secondly, we currently formally defined the syntax and structure of the composition operators. Formalizing the semantics of composition is a logical next step. Thirdly, because tool support is essential for the feasibility of the approach, we plan to implement a proof of concept. Finally, investigating the implication of the composition for other types of views, to separate other concerns and study view composition in other ADLs is also of interest.

8 ACKNOWLEDGEMENT

EMC² and the author are supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). This research was partially supported by European Commission as part of the grant IST-2-004349: European Network of Excellence on Aspect- Oriented Software Development (AOSD-Europe), 2004-2008. Thanks to the reviewers, Danny Weyns, Alessandro Garcia, Steven Op de Beeck, Alexander Helleboogh and Hilke Boucké for the discussions about this paper.

References

- [1] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.
- [2] C. Atkinson and T. Kühne. Aspect-oriented development with stratified frameworks. *IEEE Software*, 20(1):81–89, 2003.
- [3] E. Baniassad, P. C. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan. Discovering early aspects. *IEEE Softw.*, 23(1):61–70, 2006.
- [4] L. Bass, P. Clements, and R. Kazman. *Software Architectures in Practice (Second Edition)*. Addison-Wesley, 2003.
- [5] T. Batista, C. Chavez, A. Garcia, C. Sant’Anna, U. Kulesza, A. Rashid, and F. Filho. Reflections on architectural connection: Seven issues on aspects and adls. In *Workshop on Early Aspects held at ICSE’06*, 2006.
- [6] N. Boucké, T. Holvoet, T. Lefever, R. Sempels, K. Schelfhout, D. Weyns, and J. Wielemans. Applying the Architecture Tradeoff Analysis Method (ATAM) to an industrial multi-agent system application. Technical Report CW431, Dept. of Computer Science, KULeuven, 2005.
- [7] N. Boucké, D. Weyns, and T. Holvoet. Experiences with Theme/UML for architectural design of a multiagent system. In *Multiagent Systems and Software Architecture, Proceedings of the Special Track at Net.ObjectDays 2006*, pages 87–110. Net.ObjectDays, K.U.Leuven, 2006.
- [8] N. Boucké, D. Weyns, K. Schelfhout, and T. Holvoet. Applying the ATAM to an architecture for decentralized control of a transportation system. In *Quality of Software Architectures conference (QoSA)*, volume LNCS 4214, 2006.
- [9] H. V. Brussel, J. Weyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37, 1998.
- [10] G. Cabri, L. Leonardi, and F. Zambonelli. Modeling role-based interactions for agents. In *The Workshop on Agent-oriented methodologies at OOPSLA*, 2002.
- [11] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters. Semantics-based composition for aspect-oriented requirements engineering. In *Aspect Oriented Software Development conf.*, 2007.

- [12] S. Clarke. Extending standard uml with model composition semantics. *Science of Computer Programming*, 44(1):71–100, 2002.
- [13] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design*. Addison-Wesley, 2005.
- [14] S. Clarke and R. J. Walker. Composition patterns: an approach to designing reusable aspects. In *Proceedings of the International Conference on Software Engineering*, pages 5–14, 2001.
- [15] P. Clements, F. Bachman, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures, Views and Beyond*. Addison Wesley, 2003.
- [16] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley Publishing Comp., 2002.
- [17] C. Cuesta, M. Romay, P. Fuente, and M. Barrio-Solorzano. Architectural aspects of architectural aspects. In *European Workshop on Software Architecture (EWSA)*, volume LNCS 3527, pages 247–262, 2005.
- [18] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An infrastructure for the rapid development of xml-based architecture description languages. In *Proceedings of the 24th International Conference on Software Engineering (ICSE2002)*, 2002.
- [19] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, .14(2):199–245, 2005.
- [20] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7), 2002.
- [21] Eclipse. Aspectj project. www.eclipse.org/aspectj/.
- [22] Egemin. Egemin website. www.egemin.com.
- [23] Egemin and DistriNet. Emc²: Egemin modular controls concept. <http://emc2.egemin.com>.
- [24] A. Garcia, C. Chavez, T. Batista, C. Sant’anna, U. Kulesza, A. Rashid, and C. Lucena. On the modular representation of architectural aspects. In *Proc. of the European Workshop on Software Architecture*, 2006.
- [25] A. Garcia, C. Chavez, and R. Choren. Enhancing agent-oriented models with aspects. In *Proceedings of the ACM Fifth International Joint Conference on Autonomous Agents & Multi Agent Systems*, 2006.
- [26] A. Garcia, U. Kulesza, and C. Lucena. Aspectizing multi-agent systems: From architecture to implementation. *Software Engineering for Multi-Agent Systems III*, LNCS 3390:121–143, December 2004.

- [27] D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In *Foundations of Component-Based Systems*. Cambridge University Press, 2000.
- [28] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, Singapore, 1993. World Scientific Publishing Company.
- [29] Z. Huzar, L. Kuzniarz, G. Reggio, and J. L. Sourrouille. Consistency problems in uml-based software development. In *UML 2004 Satellite Activities*, volume LNCS, 2005.
- [30] IEEE. Recommended practice for architectural description of software-intensive systems (ansi/ieee-std-1471), September 2000.
- [31] ISR, Institute for Software Research. Archstudio 4.0 tool set for the xadl language. <http://www.isr.uci.edu/projects/archstudio/>.
- [32] M. A. Jackson. Some complexities in computer-based systems and their implications for system development. In *Proceedings of CompEuro90*. IEEE Computer Society Press, 1990.
- [33] R. Kazman, M. Klein, and P. Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, SEI, Carnegie Mellon University, 2000.
- [34] E. A. Kendall. Role model designs and implementations with aspect-oriented programming. In *OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 353–369, New York, NY, USA, 1999. ACM Press.
- [35] E. A. Kendall. Role modelling for agent systems analysis, design and implementation. *IEEE Concurrency*, 8:34–41, 2000.
- [36] S. Kojarski and D. H. Lorenz. Modeling aspect mechanisms: A top-down approach. In *In Proc. Int. Conf. on Software Engineering*, pages 212–221, 2006.
- [37] D. C. Luckham. Rapide: A language and toolset for simulation of distributed systems by partial orderings of events. In *DIMACS Partial Order Methods Workshop IV*, July 1996. 25 pages.
- [38] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.
- [39] A. Moreira, A. Rashid, and J. Araujo. Multi-dimensional separation of concerns in requirements engineering. In *The 13th International Conference on Requirements Engineering (RE'05)*, pages 285–296, 2005.
- [40] B. Nuseibeh, J. Kramer, and A. Finkelstein. Viewpoints: meaningful relationships are difficult! In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 676–681, Washington, DC, USA, 2003. IEEE Computer Society.

- [41] H. Ossher and P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluwer, 2000.
- [42] J. Perez, I. Ramos, J. Jaen, P. Letelier, and E. Navarro. Prisma: Towards quality, aspect oriented and dynamic software architectures. In *Int. Conf. On Quality Software*, 2003.
- [43] M. Pinto, L. Fuentes, and J. M. Troya. A dynamic component and aspect-oriented platform. *Computing Journal*, 48(4):401–420, 2005.
- [44] C. Pons, R. Giandini, and G. Baum. Dependency relations between models in the unified process. 2000.
- [45] N. Rozanski and E. Woods. *Software Systems Architecture*. Addison Wesley, 2005.
- [46] O. Shehory. Architectural properties of multiagent systems. Technical Report CMU-RI-TR-98-28, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1998.
- [47] L. Silva. An aspect-oriented approach to model requirements. In *Proc. of the Doctoral Consortium on Requirements Engineering (with RE conference)*, 2005.
- [48] Software Engineering Institute. The sae aadl standard info site. <http://www.aadl.info/>.
- [49] P. Tarr, H. Ossher, W. Harrison, and S. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *Int. Conf. on Software Engineering*, pages 107–119, 1999.
- [50] W3C. Xml schema. <http://www.w3.org/XML/Schema>.
- [51] D. Weyns. *An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems*. PhD thesis, KULeuven, Belgium, 2006.
- [52] D. Weyns and T. Holvoet. Multiagent systems and software architecture. In *Special Track on Multiagent Systems and Software Architecture, Net.ObjectDays*, 2006.
- [53] D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems: State-of-the-art and research challenges. In *Revised papers of the E4MAS workshop at AAMAS'04*, volume LNCS 3374, 2005.
- [54] D. Weyns, K. Schelfhout, T. Holvoet, and T. Lefever. Decentralized control of E'GV transportation systems. In *International Conference on Autonomous Agents and Multi-Agent Systems, Industry Track*, pages 25–29, July 2005.