ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SODA

Societies in Open Distributed Agent spaces

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SODA

Societies in Open Distributed Agent spaces

Date:	22nd December 2010
Version:	First Draft
Pages:	
Responsible:	Ambra Molesini
Authors:	Ambra Molesini
	Andrea Omicini
Contact:	ambra.molesini@unibo.it

Contents

1 Introduction **2**

1.1 The SODA Process Lifecycle 3
1.2 The Meta-model 4
1.3 The Layering 10
1.3.1 Process roles 12
1.3.2 Activity Details 12
1.3.3 Work products 13

2 Phases of the SODA process **17**

2.1 The Requirements Analysis 17
2.1.1 Process roles 17
2.1.2 Activity Details 17
2.1.3 Work products 20
2.2 The Analysis 23
2.2.1 Process roles 23
2.2.2 Activity Details 26
2.2.3 Work products 28
2.3 The Architectural Design 32
2.3.1 Process roles 35
2.3.2 Activity Details 35
2.3.3 Work products 39
2.4 The Detailed Design 42
2.4.1 Process roles 43
2.4.2 Activity Details 43
2.4.3 Work products 50

3 Work products dependencies **57**

Bibliography **59**

1 Introduction

SODA (Societies in Open and Distributed Agent spaces) [1, 2, 3, 5] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts (A&A) meta-model [6], and introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the analysis and design process. Since its first version [5], SODA is not concerned with *intra-agent* issues: designing a multi-agent system with SODA leads to defining agents in terms of their required observable behaviour and their role in the multi-agent system. Then, whichever methodology one may choose to define the agent structure and inner functionality, it could be easily used in conjunction with SODA. SODA concentrated on *inter-agent* issues, like the engineering of societies and environment for multi-agent systems.

SODA abstractions are logically divided into three categories: *i)* the abstractions for modelling/designing the system's active part (task, role, agent, etc.); *ii)* those for the reactive part (function, resource, artifact, etc.); and *iii)* those for interaction and organisational rules (relation, dependency, interaction, rule, etc.). In its turn, the SODA *process* is organised in two phases, each structured in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models (designs) the system exploiting a subset of the SODA abstractions: in particular, each subset always includes at least one abstraction for each of the above categories—that is, at least one abstraction for the system's active part, one for the reactive part, and another for interaction and organisational rules.

An overview of the methodology structure is shown in Figure 1.1: each step is practically described in terms of a set of relational tables, listed in the figure. SODA provides only a tabular representation of the system. Each table in the methodology has both a full name and a unique acronym, which specifies the layer (see Section 1.3) where the table belongs (in round brackets) and the involved entities. For instance, $(L)AR_t$ refers to the Actor (specified by the "A")-Requirement (specified by the "R") table (specified by the "t") at layer L.

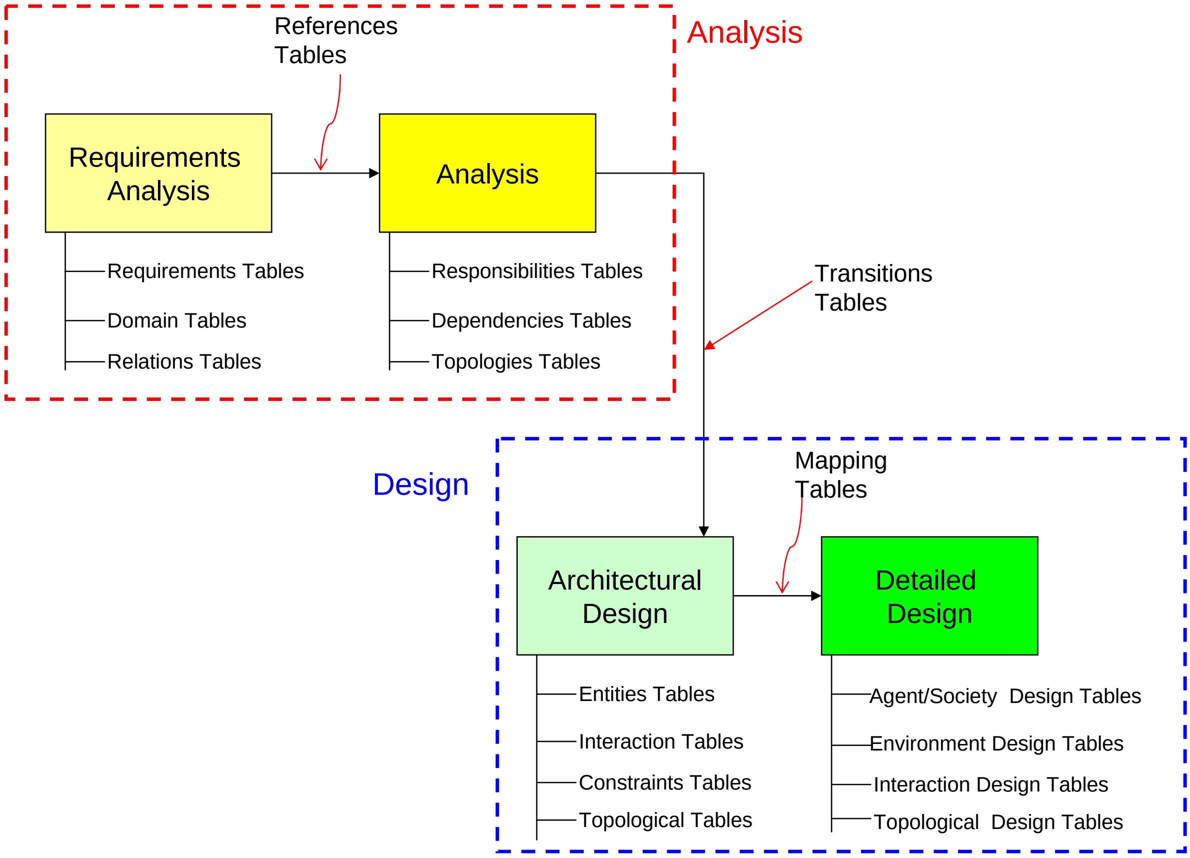

Figure 1.1: An overview of the SODA process

1.1 The SODA Process Lifecycle

SODA includes two phases each of them structure in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps.

These phases are arranged in iterative process model (see Figure 1.2):

- Requirements Analysis covers all the phases related to the Actor identification, the Requirements Elicitation and Analysis and the Analysis of the existing environment.

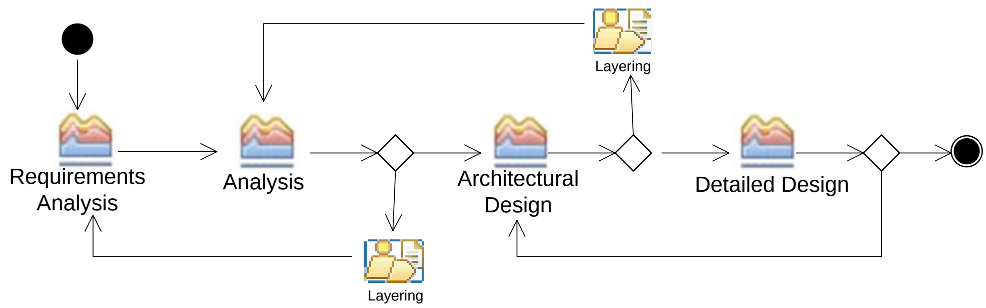

Figure 1.2: The SODA process phases

- Analysis investigates all the aspects related to the problem domain trying to understand the tasks satisfying the requirements, their connected functions, the environment topology and all the dependencies among these entities.

- Architectural Design defines a family of possible architectures for the final system.

- Detailed Design establish the best system architecture and design the environment and the system's interactions.

Each step produces several sets of relational tables each of them describes a specific MAS Meta-model Element (MMMElement) and its relationships with the other MMMElements.

The details of each step will be discussed in the following section.

1.2 The Meta-model

The meta-model that represents the abstract entities adopted by SODA is depicted in Figure 1.3.

Requirements Analysis. Several abstract entities are introduced for requirement modelling (see Figure 1.3 "requirements analysis" part): in particular, *Requirement* and *Actor* are used for modelling the customers' requirements and the requirement sources, respectively, while the *ExternalEnvironment* notion is used as a container of the *LegacySystems* that represent the legacy resources of the environment. The relationships between requirements and legacy systems are then modelled in terms of suitable *Relation* entities.

Analysis. The Analysis step expresses the abstract requirement representation in terms of more concrete entities such as *Tasks* and *Functions* (see

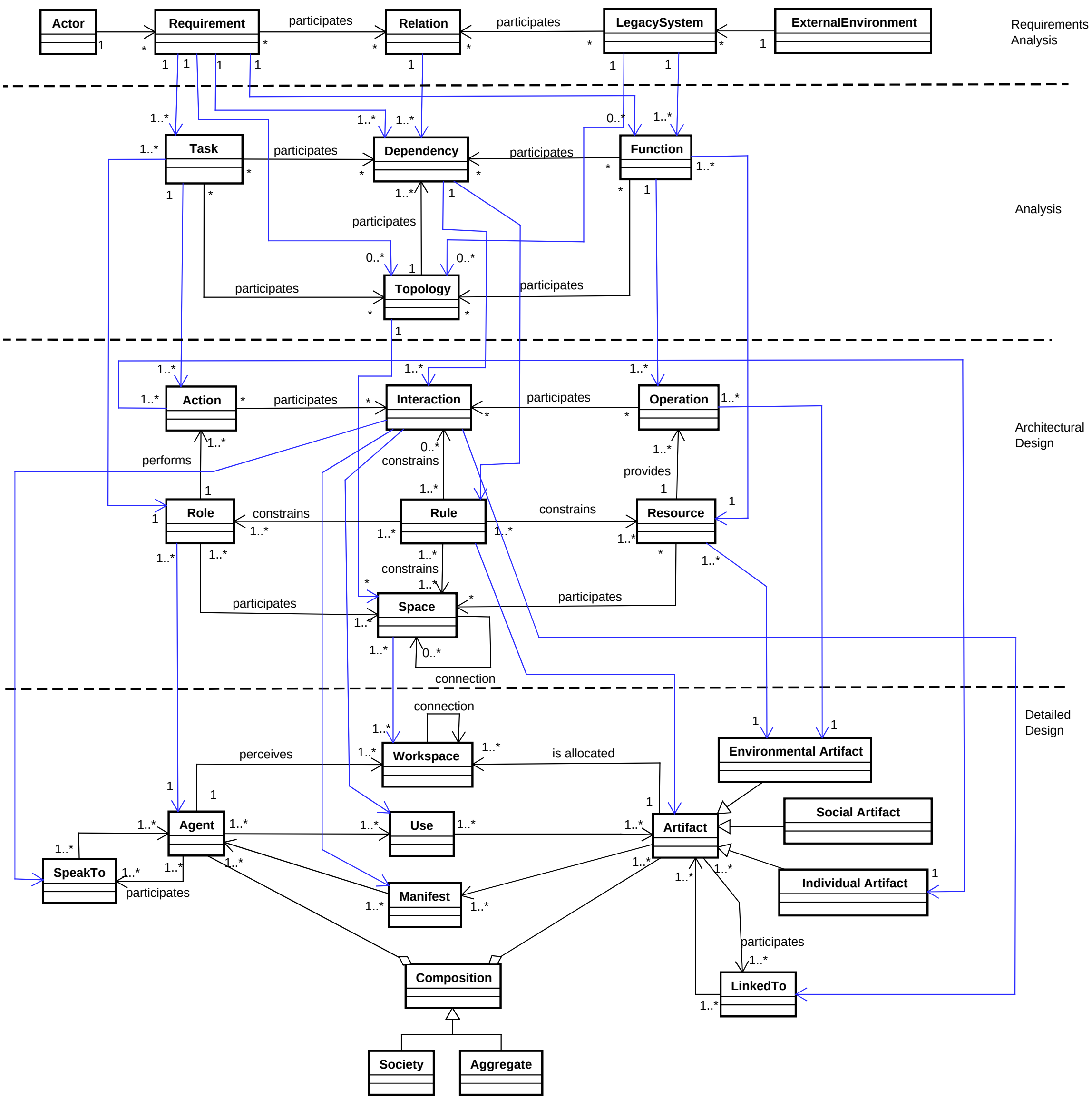

Figure 1.3: The SODA Meta-model

Figure 1.3, “analysis” part). Tasks are activities requiring one or more competences, while functions are reactive activities aimed at supporting tasks. The relations highlighted in the previous step are now the starting point for the definition of *Dependencies* (interactions, constraints, etc.) among the abstract entities. The structure of the environment is also modelled in terms of *Topologies*, i.e. topological constraints over the environment.

Topologies are often derived from functions, but can also constrain / affect task achievement.

Architectural Design. The main goal of this stage is to assign responsibilities of achieving tasks to *Roles*, and responsibilities of providing functions to *Resources* (see Figure 1.3, “architectural design” part). To this end, roles should be able to perform *Actions*, and resources should be able to execute *Operations* providing one or more functions. The dependencies identified in the previous phase become here *Interactions* and *Tules*. Interactions represent the acts of the interaction among roles, among resources and between roles and resources; rules, instead, enable and bound the entities' behaviour. Finally, the topology constraints lead to the definition of *Spaces*, i.e. conceptual places structuring the environment.

Detailed Design. The active and passive parts are expressed in the Detailed Design in terms of *Agents*, agent *Societies*, *Artifacts* and *Aggregate*, (see Figure 1.3 “detailed design” part). Agents are intended here as autonomous entities able to play several roles, and the resources identified in the previous step are now mapped onto suitable artifacts. In the meta-model the artifact is reported specifying its “type” derived from the following taxonomy [7]:

- *Individual artifact* handles the interaction of a single agent within a MAS, and essentially works as a mediator between the agent and the MAS itself. Since they can be used to shape of admissible interactions of individual agents in MAS, individual artifacts play an essential role in engineering both organisational and security concerns in MAS.

- *Environmental artifact* brings an external resource within a MAS, by mediating agent actions and perceptions over resources. As such, environmental artifacts play an essential role in enabling, disciplining and governing the interaction between agents and MAS environment.

- *Social artifact* rules social interactions within a MAS—even though indirectly, since it technically mediates interactions between individual, environmental, and possibly other social artifacts. Social artifacts in SODA play the role of the coordination artifacts that embody the rules around which societies of agents can be built.

In SODA a society can be seen as a group of interacting agents and artifacts when its overall behaviour is essentially an autonomous, proactive one; it can be seen an aggregate when its overall behaviour is essentially a functional, reactive one.

The *workspaces* take the form of an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, and agents can dynamically enter (join) or exit workspaces.

Finally, the *Use*, *Manifest*, *SpeakTo* and *LinkedTo* concepts represent the different forms of interaction supported by SODA.

Table 1.1: The SODA entities definitions

Concepts	**Definition**	**Step**
Actor	System's stakeholder	Requirements Analysis
Requirement	Service that the stakeholder requires from a system and the constraints under which it operates and is developed	Requirements Analysis
LegacySystem	Represent the legacy resources of the environment	Requirements Analysis
External Environment	Represent the legacy world in which the new system will execute	Requirements Analysis
Relation	Represent all the possible ties among the entities of the Requirements Analysis	Requirements Analysis
Task	Represent a computation that contributes to the satisfaction of a specific requirements.	Analysis
Function	Reactive activity aimed at supporting task (or goal)	Analysis
Topology	Topological constraints over the environment. Topologies are often derived from legacy-systems and requirements, but also functions and tasks could induct some topological constraints	Analysis

Dependency	Represents interactions, constraints, and kind of relationships among the abstract entities	Analysis
Role	Represent an entity responsible to accomplish some tasks	Architectural Design
Action	Represent a computation that changes the environment in order to meet roles design objectives	Architectural Design
Resource	Entity that provides functions	Architectural Design
Operation	Finer-grained functionality provided by resource in order to achieve a function	Architectural Design
Space	Conceptual places structuring the environment	Architectural Design
Rule	Enables and bound the entities' behaviour. Such entities are roles, resources, interactions, and spaces	Architectural Design
Interaction	Represent the acts of the interaction among roles, among resources and between roles and resources	Architectural Design
Composition	Represent a generic composition of agents and artifact without specifying the overall "nature"	Detailed Design
Agent	Represent pro-active components of the systems, encapsulating the autonomous execution of some kind of activities inside some sort of environment	Detailed Design

Society	A group of interacting agents and artifacts when its overall behaviour is essentially an autonomous, proactive one	Detailed Design
Artifact	Represent passive components of the systems such as resources and media that are intentionally constructed, shared, manipulated and used by agents to support their activities, either cooperatively or competitively	Detailed Design
Individual Artifact	Mediator between the agent and the MAS	Detailed Design
Social Artifact	Ruling social interactions within a MAS	Detailed Design
Environmental Artifact	disciplining and governing the interaction between agents and MAS environment	Detailed Design
Aggregate	A group of interacting agents and artifacts when its overall behaviour is essentially a functional, reactive one	Detailed Design
Workspace	Represent conceptual containers of agents and artifacts, useful for defining the topology for the environment and providing a way to define a notion of locality	Detailed Design
Use	The act of interaction between agent and artifact: agent uses artifact	Detailed Design

Manifest	The act of interaction between artifact and agent: artifact manifests itself to agent	Detailed Design
SpeakTo	The act of interaction among agents: agent speaks with another agent	Detailed Design
LinkedTo	The act of "interaction" among artifact: artifact is linked to another artifact	Detailed Design

1.3 The Layering

Complexity is inherent in real-life systems. While *modelling* complex systems and understanding their behaviour and dynamics is the most relevant concern in many areas, such as economics, biology, or social sciences, in the software systems also the complexity of *construction* becomes an interesting challenge. An integral part of a system development methodology must therefore be a set of tools for controlling and managing this complexity. So, soda has introduced a tool in order to deal with the complexity of the system representation: the system is represented as composed by different layers of abstraction and it is possible to move from one layer to another layer by means of a *layering operation.*

This tool in SODA is called "Layering" and it is represented as a *capability pattern* [4], i.e., a reusable portion of the process, as shown in Figure 1.4 where the layering process is detailed. In particular, the layering presents two different functionalities: *(i)* the selection of a specific layer for refining / completing the abstractions models in the methodology process (Select Layer activity), and *(ii)* the creation of a new layer in the system by in-zooming – i.e., increasing the system detail – or out-zooming – i.e., increasing the system abstraction – activities. In latter case, the layering process terminates with the projection activity needed to project the abstractions from one layer to another "as they are", so as to maintain the consistency in each layer.

In general, when working with SODA, the starting layer, called *core layer*, is labelled with "C" and is always *complete*—that is, it contains all the entities required to fully describe a given abstract layer. Any other layer contains just *i)* the entities that have been possibly (in/out-) zoomed from another layer, as well as *ii)* the entities possibly projected "as they are" from other layers: so, in general, these layers are not necessarily complete—though of course they might be so.

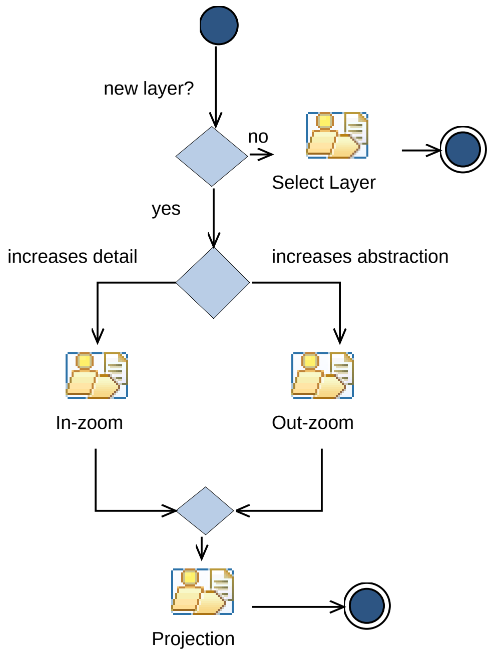

Figure 1.4: The Layering process

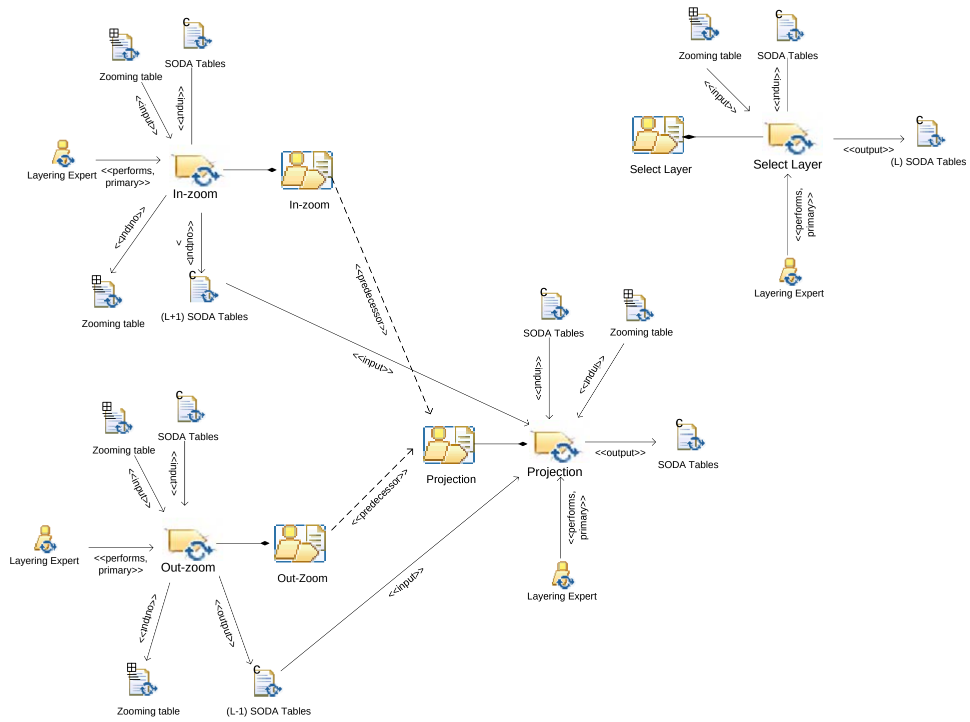

Figure 1.5: The Layering flow of activities, roles and work products

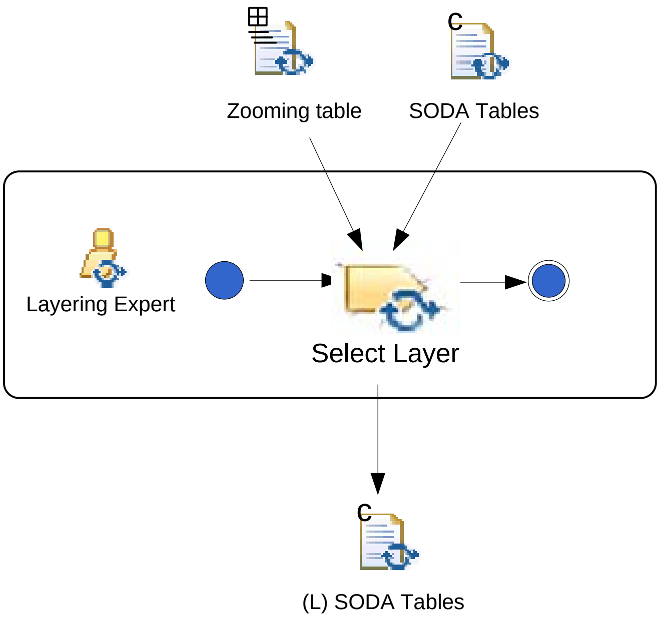

Figure 1.6: The flow of task in the Select Layer activity

1.3.1 Process roles

One role is involved in the Layering pattern: the Layering Expert. Layering Expert is responsible for:

- selecting the specific abstraction layer
- in-zooming or out-zooming the system by creating or modifying the Zooming table
- projecting the necessary entities in the new created layer by partially filling all the tables

1.3.2 Activity Details

Select Layer activity

The flow of tasks inside this activity is reported in Figure 1.6 and the tasks are detailed in the following table.

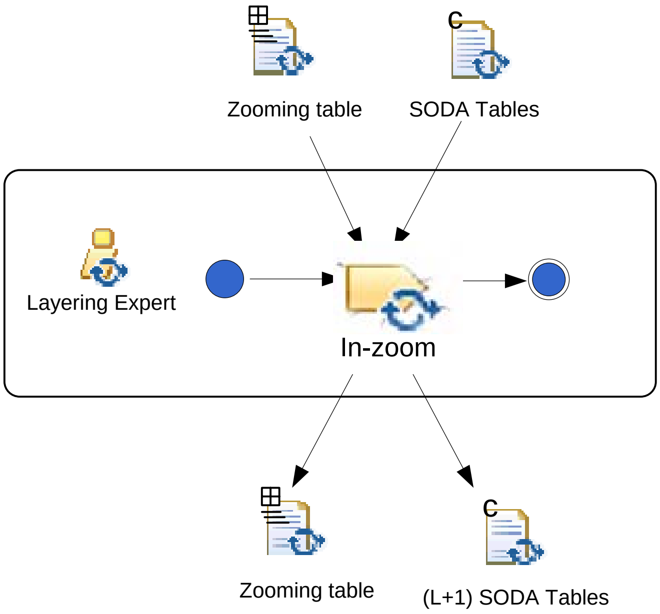

Figure 1.7: The flow of task in the In-zoom activity

In-zoom activity

The flow of tasks inside this activity is reported in Figure 1.7 and the tasks are detailed in the following table.

Out-zoom activity

The flow of tasks inside this activity is reported in Figure 1.8 and the tasks are detailed in the following table.

Projection activity

The flow of tasks inside this activity is reported in Figure 1.9 and the tasks are detailed in the following table.

1.3.3 Work products

The Layering generates one work product (the Zooming table). Its relationships with the MMMElements are described in Figure 1.10.

This diagram represents the Layering in terms of the Work Product and its relationships with the SODA meta-model (Section 1.2) elements. Each MMMElement is represented using an UML class icon (yellow filled) and, in

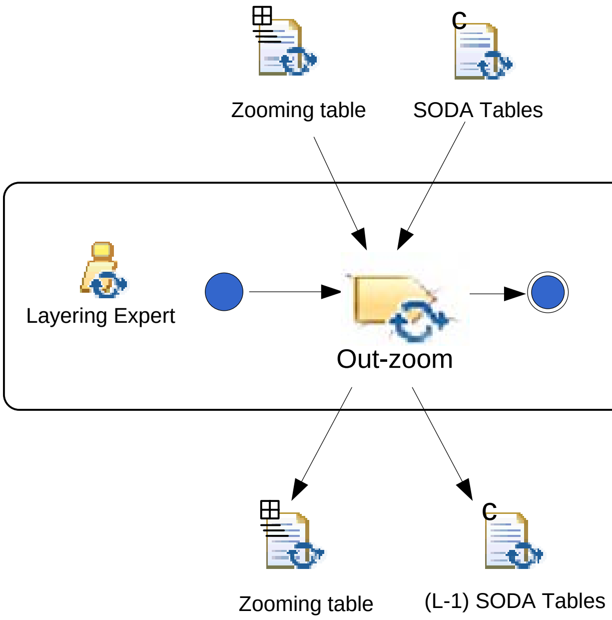

Figure 1.8: The flow of task in the Out-zoom activity

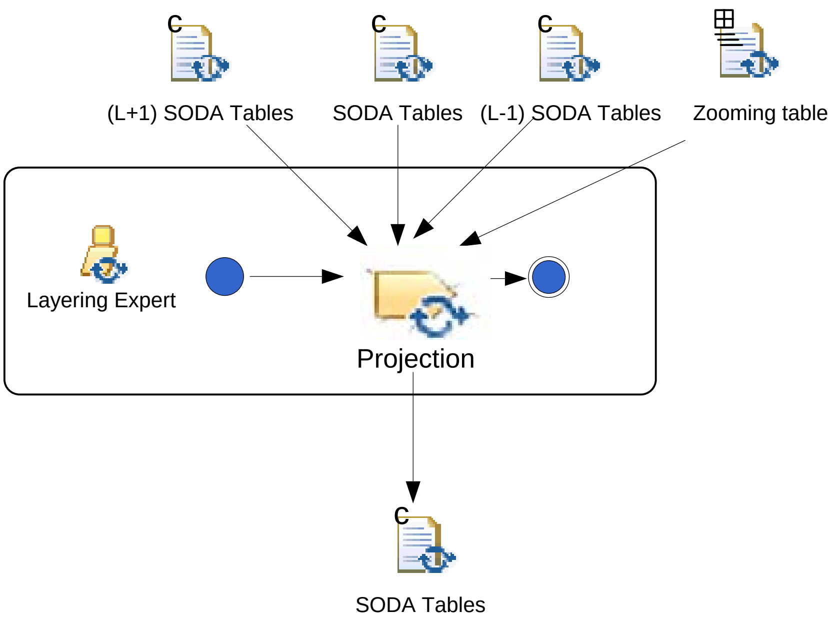

Figure 1.9: The flow of task in the Projection activity

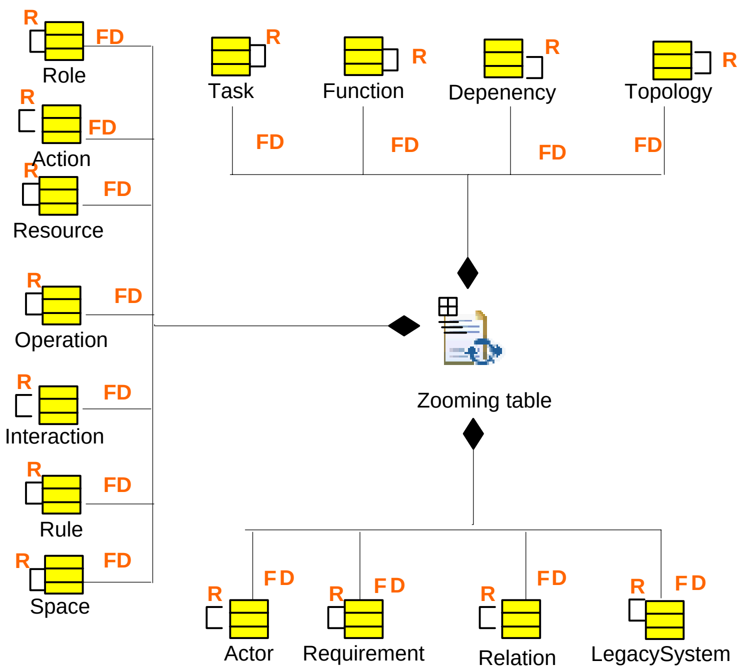

Figure 1.10: The Layering work products

the documents, such elements can be Defined, reFined, Quoted, Related or Relationship Quoted as described below:

- *defined* (D label): this means that the element is introduced for the first time in the design in this artefact (the MMMElement is instantiated in this artefact);
- *reFined* (F label): this means that the MMMElement is refined in the work product (for instance by means of attribute definition);
- *related* (R label): this means that an already defined element is related to another or from a different point of view that one of the MAS meta-model relationships is instantiated in the document;
- *quoted* (Q label): this means that the element has been already defined and it is reported in this artefact only to complete its structure but no work has to be done on it;
- *relationship quoted* (RQ label): this means that the relationship is reported in the work product but it has been defined in another part of the process

The Layering is represented by means of a Zooming Table ($(C)Z_t$)(Figure 1.11) The Zooming table formalises the in-zoom of a layer into the more detailed layer; of course, the same table can be used to represent the dual out-zoom process.

Layer L	Layer L+1
out-zoomed entity	*in-zoomed entities*

Figure 1.11: $(L)Z_t$

2 Phases of the SODA process

2.1 The Requirements Analysis

The goal of Requirements Analysis is the characterisation of both the customers' requirements and the legacy systems with which the system should interact, as well as to highlight the relationships among requirements and legacy systems. Figure 2.1 presents the Requirements Analysis process, while Figure 2.2 presents the flow of activities, the involved roles and the work products

2.1.1 Process roles

Three roles are involved in the Requirements Analysis: the Requirement Analyst, the Environment Analyst and the Domain Analyst.

Requirement Analyst

Requirement Analyst is responsible for:

Environment Analyst

Environment Analyst is responsible for:

Domain Expert

He supports the Requirement Analyst and the Environment Analyst during the description of the application domain.

2.1.2 Activity Details

For the details about the different Layering activities please refer Section 1.3.

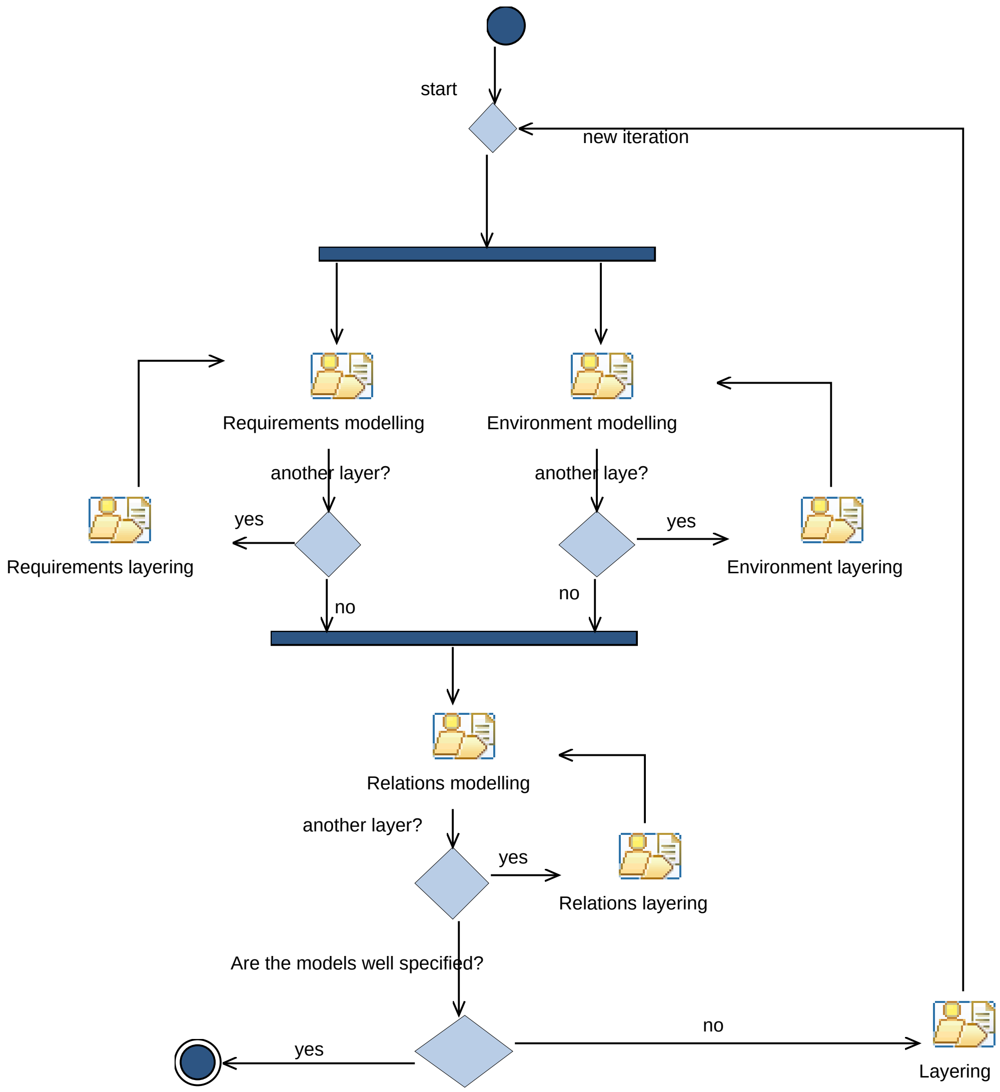

Figure 2.1: The Requirements Analysis process

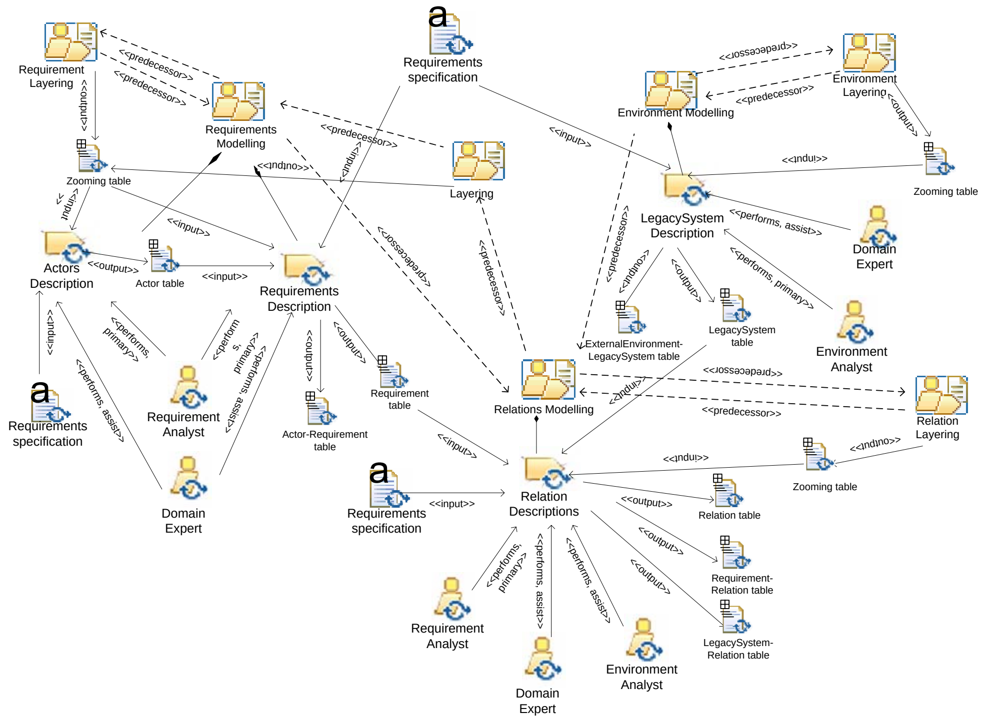

Figure 2.2: The Requirements Analysis flow of activities, roles and work products

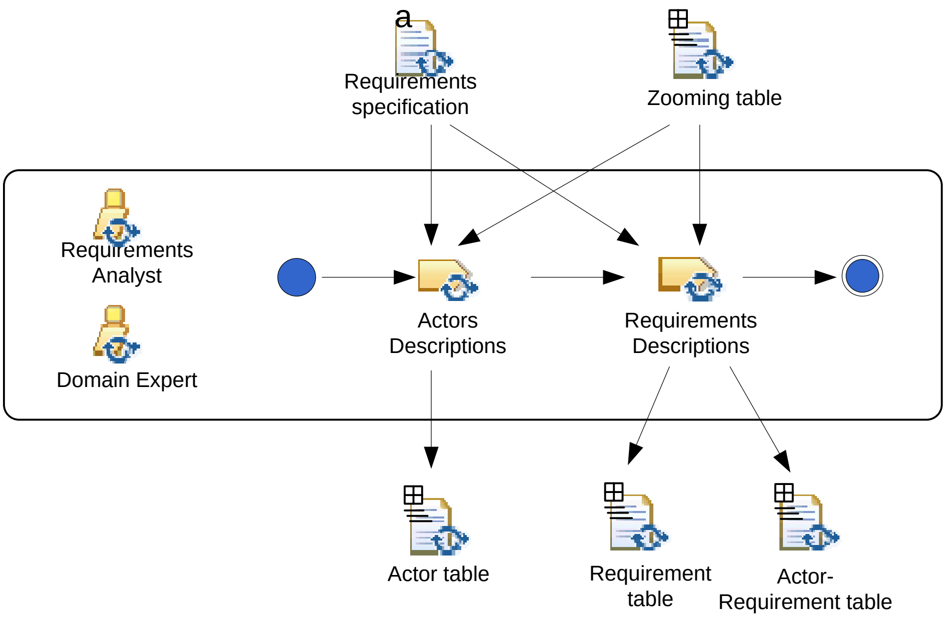

Figure 2.3: The flow of task in the Requirements Modelling activity

Requirements Modelling activity

The flow of tasks inside this activity is reported in Figure 2.3 and the tasks are detailed in the following table.

Environment Modelling activity

The flow of tasks inside this activity is reported in Figure 2.4 and the tasks are detailed in the following table.

Relations Modelling activity

The flow of tasks inside this activity is reported in Figure 2.5 and the tasks are detailed in the following table.

2.1.3 Work products

The Requirements Analysis step consists of three sets of tables: Requirements Tables, Domain Tables and Relations Tables. Their relationships with the MMMElements are described in Figure 2.6.

This diagram represents the Layering in terms of the Work Product and its relationships with the SODA meta-model (Section 1.2) elements.

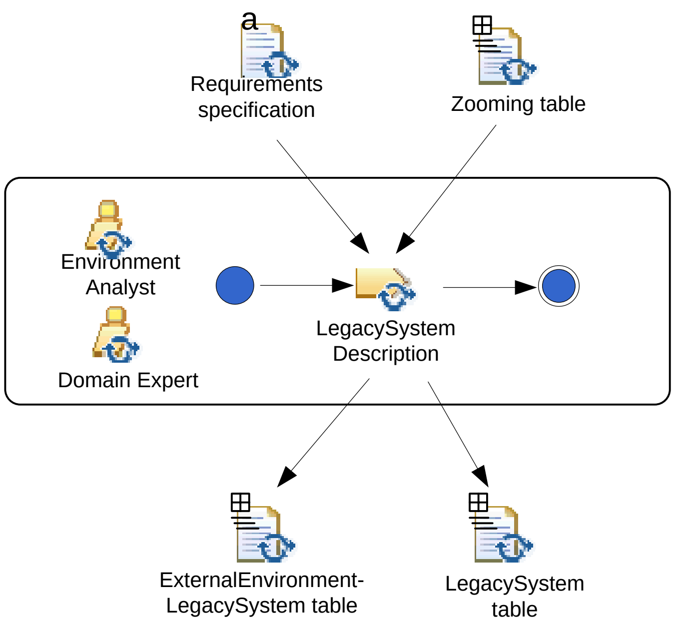

Figure 2.4: The flow of task in the Environment Modelling activity

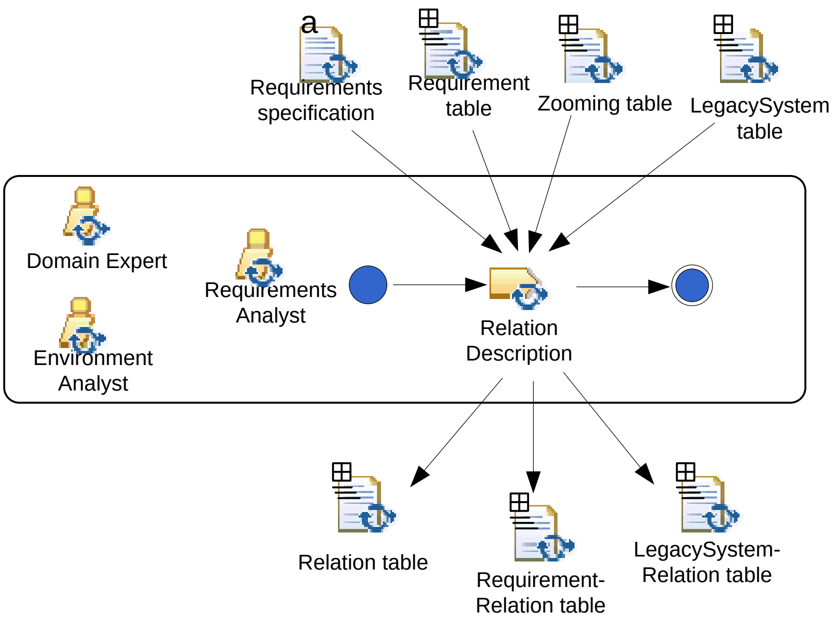

Figure 2.5: The flow of task in the Relations Modelling activity

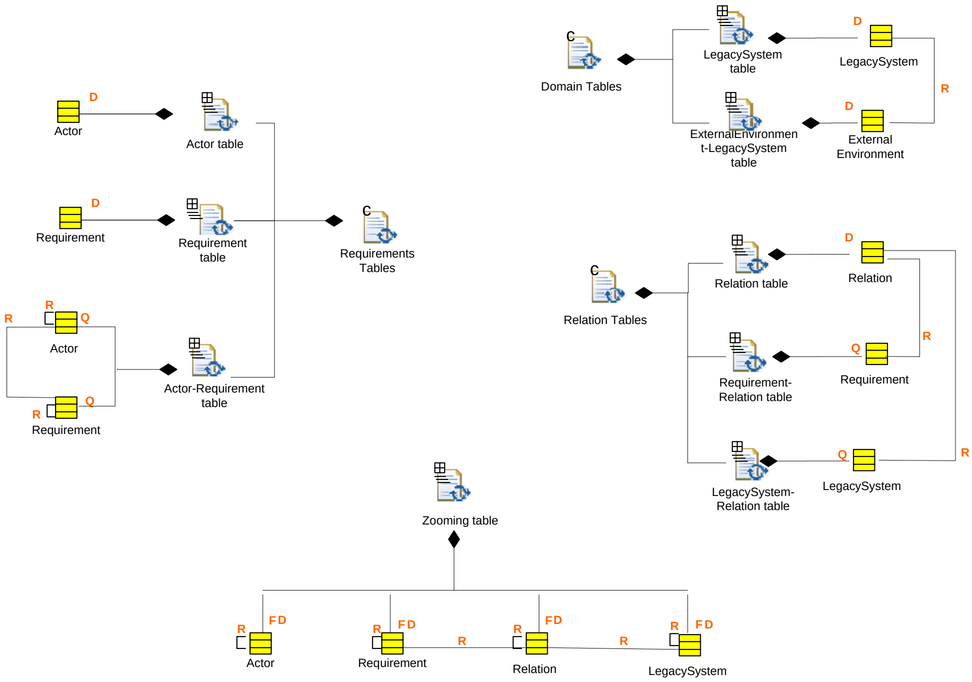

Figure 2.6: The Requirement Analysis work products

Details of the tables

Requirements Tables (Figure 2.7) define the abstract entities tied to the concept of "requirement": in particular, the Actor table ($(L)Ac_t$) describes each single actor, the Actor-Requirement table ($(L)AR_t$) specifies the collection of the requirements associated to each actor, while the Requirement table ($(L)Re_t$) describes each single requirement.

Actor	Description
actor name	*actor description*

Requirement	Description
requirement name	*requirement description*

Actor	Requirement
actor name	*requirement names*

Figure 2.7: Requirements Tables, in top-down order: $(L)Ac_t$, $(L)Re_t$, $(L)AR_t$

Domain Tables (Figure 2.8) define the abstract entities tied to the concept of "external environment". This group of tables is composed of the ExternalEnvironment-

LegacySystem table ($(L)EELS_t$), which specifies the legacy systems associated to the external environment, and the LegacySystem table ($(L)LS_t$), which describes each single legacy system.

External-Environment	Legacy-System
external-environment name	*Legacy-System names*

Legacy-System	Description
legacy-system name	*legacy-system description*

Figure 2.8: Domain Tables, in top-down order: $(L)EELS_t$, $(L)LS_t$

Finally Relations Tables (Figure 2.9) link the abstract entities with each other. In particular, the Relation table ($(L)Rel_t$) describes all the relationships among abstract entities, while the Requirement-Relation table ($(L)RR_t$) specifies the relations where each requirement is involved, and the LegacySystem-Relation table ($(L)LSR_t$) specifies the relations where each legacy-system is involved.

Relation	Description
relation name	*relation description*

Requirement	Relation
requirement name	*relation names*

Legacy-System	Relation
legacy-system name	*relation names*

Figure 2.9: Relations Tables, in top- down order: $(L)Rel_t$, $(L)RR_t$, $(L)LSR_t$.

2.2 The Analysis

Figure 2.10 presents the Analysis process, while Figure 2.11 presents the flow of activities, the involved roles and the work products

2.2.1 Process roles

One role is involved in the Layering pattern: the System Analyst.

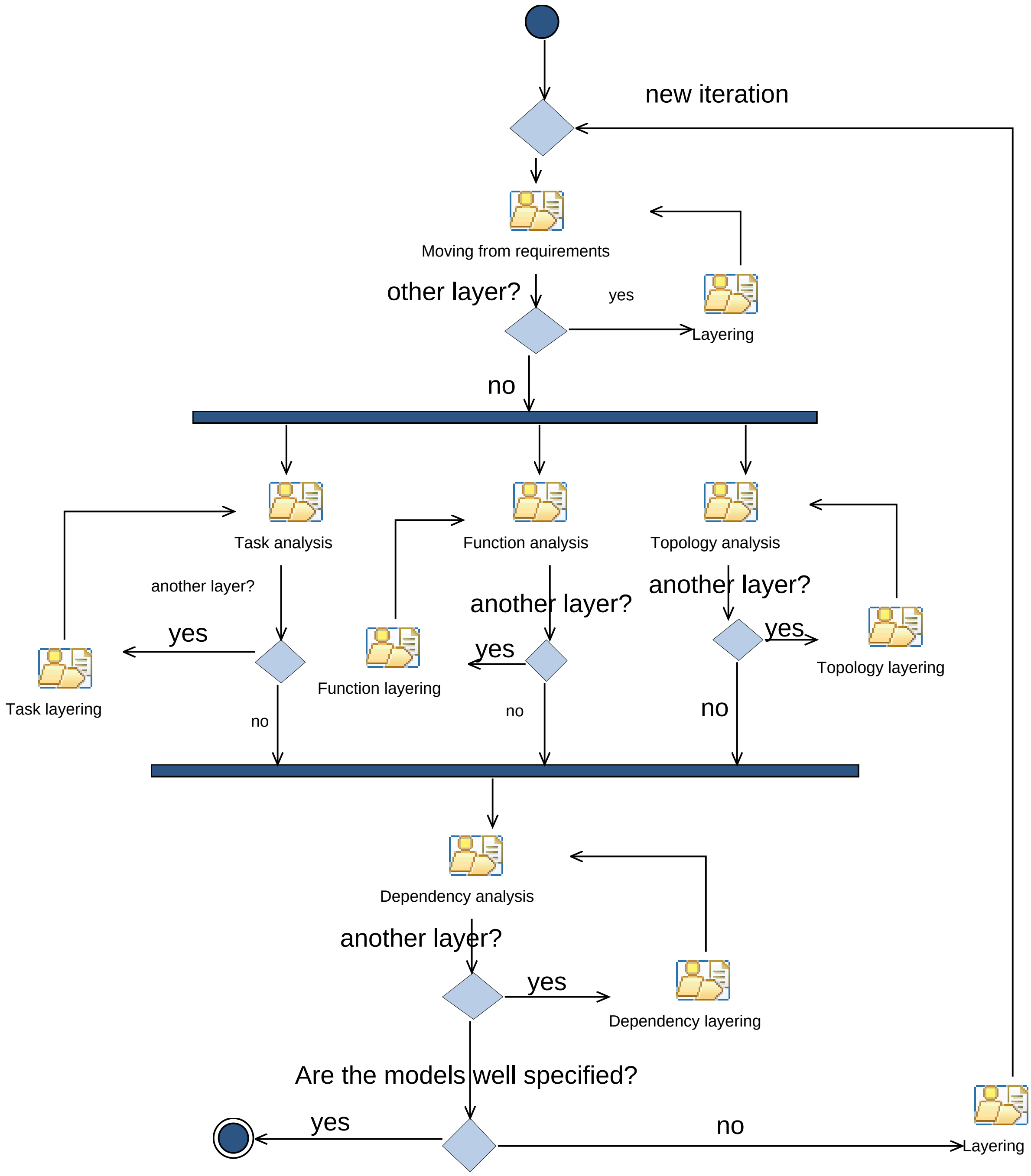

Figure 2.10: The Requirements Analysis process

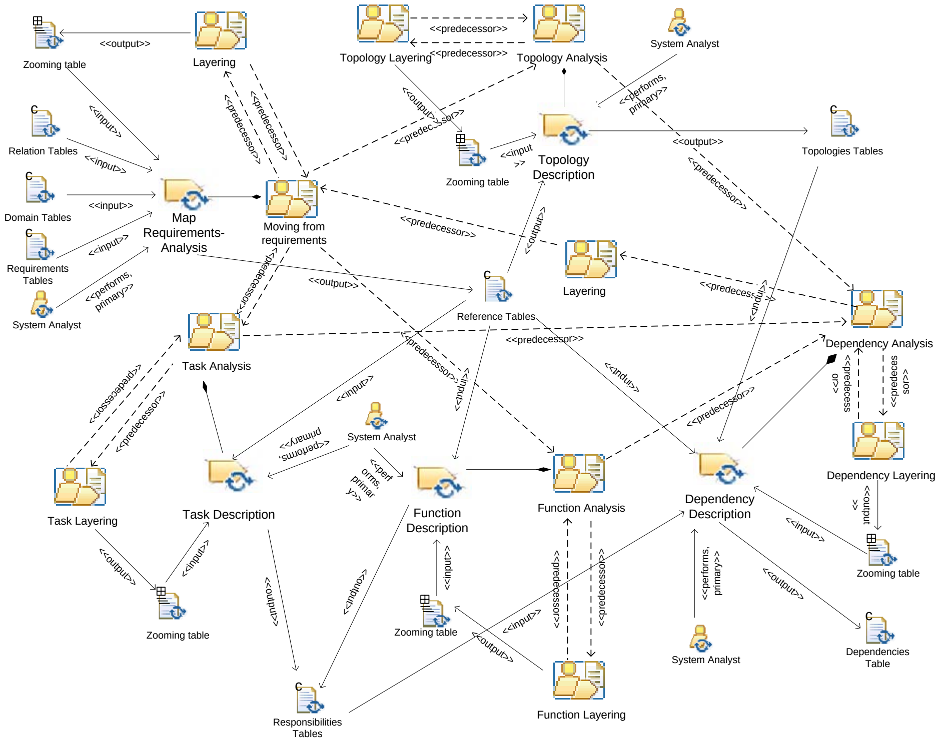

Figure 2.11: The Analysis flow of activities, roles and work products

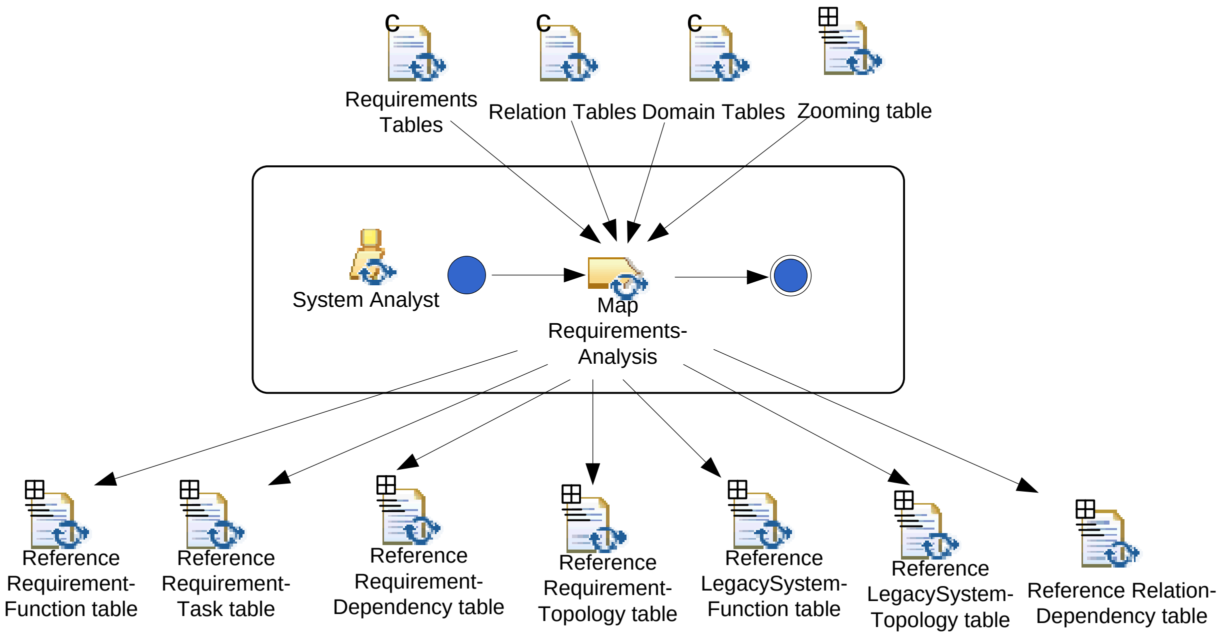

Figure 2.12: The flow of task in the Moving from requirements activity

System Analyst

System Analyst is responsible for:

2.2.2 Activity Details

For the details about the different Layering activities please refer Section 1.3.

Moving from requirements activity

The flow of tasks inside this activity is reported in Figure 2.12 and the tasks are detailed in the following table.

Task Analysis activity

The flow of tasks inside this activity is reported in Figure 2.13 and the tasks are detailed in the following table.

Function Analysis activity

The flow of tasks inside this activity is reported in Figure 2.26 and the tasks are detailed in the following table.

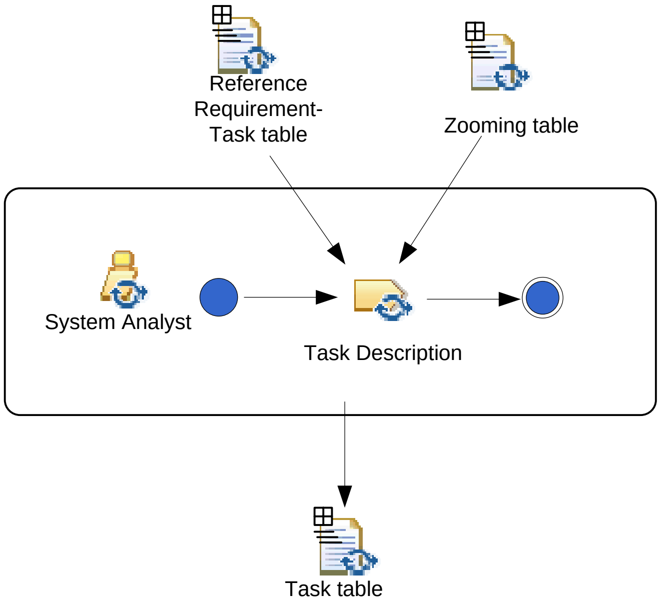

Figure 2.13: The flow of task in the Task Analysis activity

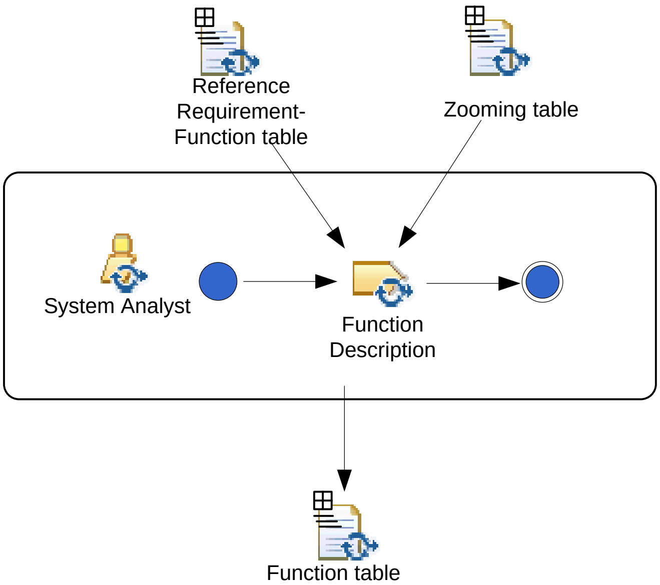

Figure 2.14: The flow of task in the Function Analysis activity

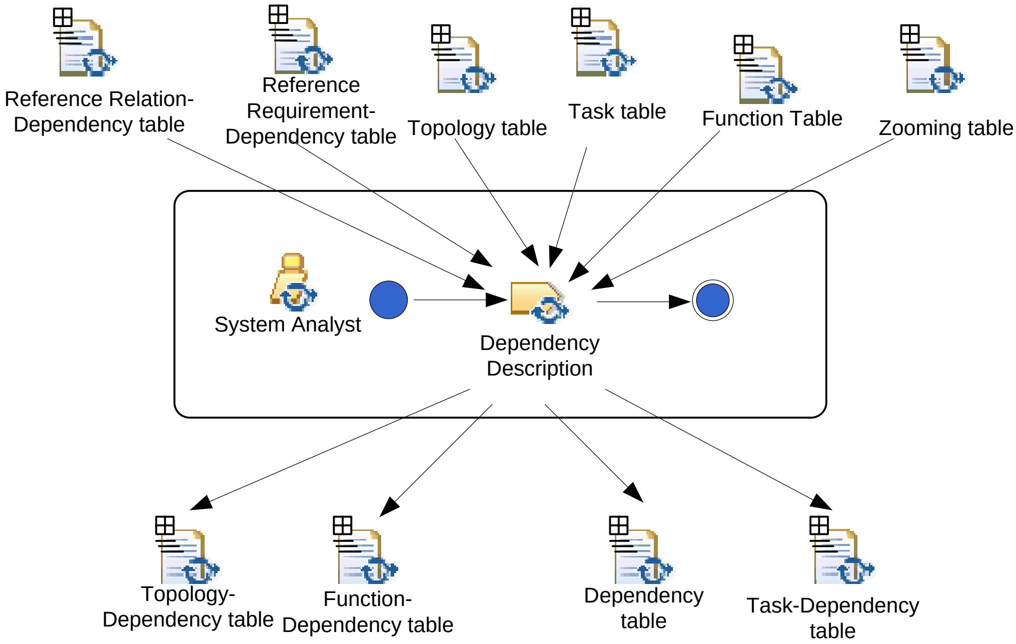

Figure 2.15: The flow of task in the Dependency Analysis activity

Dependency Analysis activity

The flow of tasks inside this activity is reported in Figure 2.15 and the tasks are detailed in the following table.

Topology Analysis activity

The flow of tasks inside this activity is reported in Figure 2.16 and the tasks are detailed in the following table.

2.2.3 Work products

The Analysis step exploits four sets of tables: Reference Tables, Responsibilities Tables, Dependencies Tables and Topologies Tables.

Their relationships with the MMMElements are described in Figure 2.17.

This diagram represents the Layering in terms of the Work Product and its relationships with the SODA meta-model (Section 1.2) elements.

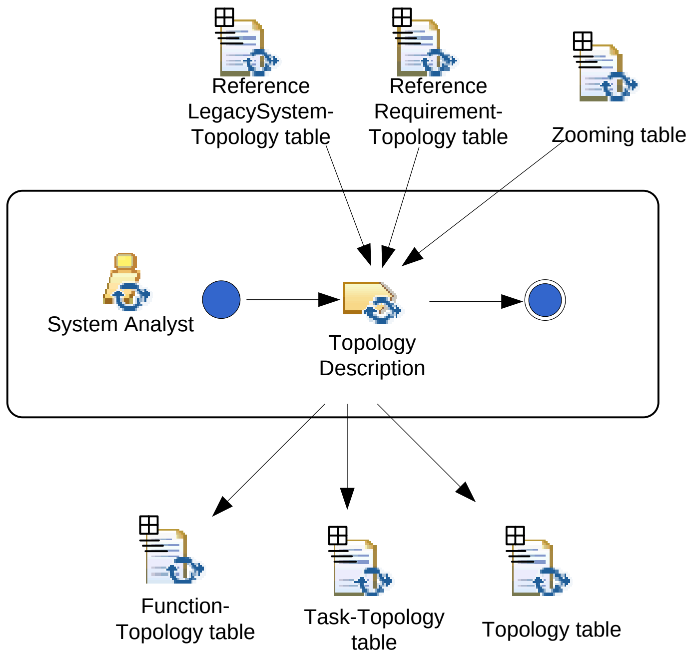

Figure 2.16: The flow of task in the Topology Analysis activity

Details of the tables

In order to move from Requirements Analysis to Analysis, the relations between the different abstractions adopted in the two steps must be precisely identified: this is done by means of the References Tables (Figure 2.18).

In particular, the Reference Requirement-Task table ($(L)RRT_t$) specifies the mapping between each requirement and the generated tasks, the Reference Requirement-Function table ($(L)RRF_t$) specifies the mapping between each requirement and the generated functions; the Reference Requirement-Topology table ($(L)RRTo_t$) specifies the mapping between each requirement and the generated topologies; the Reference Requirement-Dependency table ($(L)RReqD_t$) specifies the mapping between each requirement and the generated dependencies; the Reference LegacySystem-Function table ($(L)RLSF_t$), which specifies the mapping between each legacy-system and the corresponding functions; the Reference LegacySystem-Topology table ($(L)RLST_t$), which specifies the mapping between legacy-systems and topologies; and the Reference Relation-Dependency table ($(L)RRelD_t$), which specifies the mapping between relations and dependencies.

Responsibilities Tables (Figure 2.19) define the abstract entities tied to the concept of "responsibilities centre"—namely, tasks and functions. So,

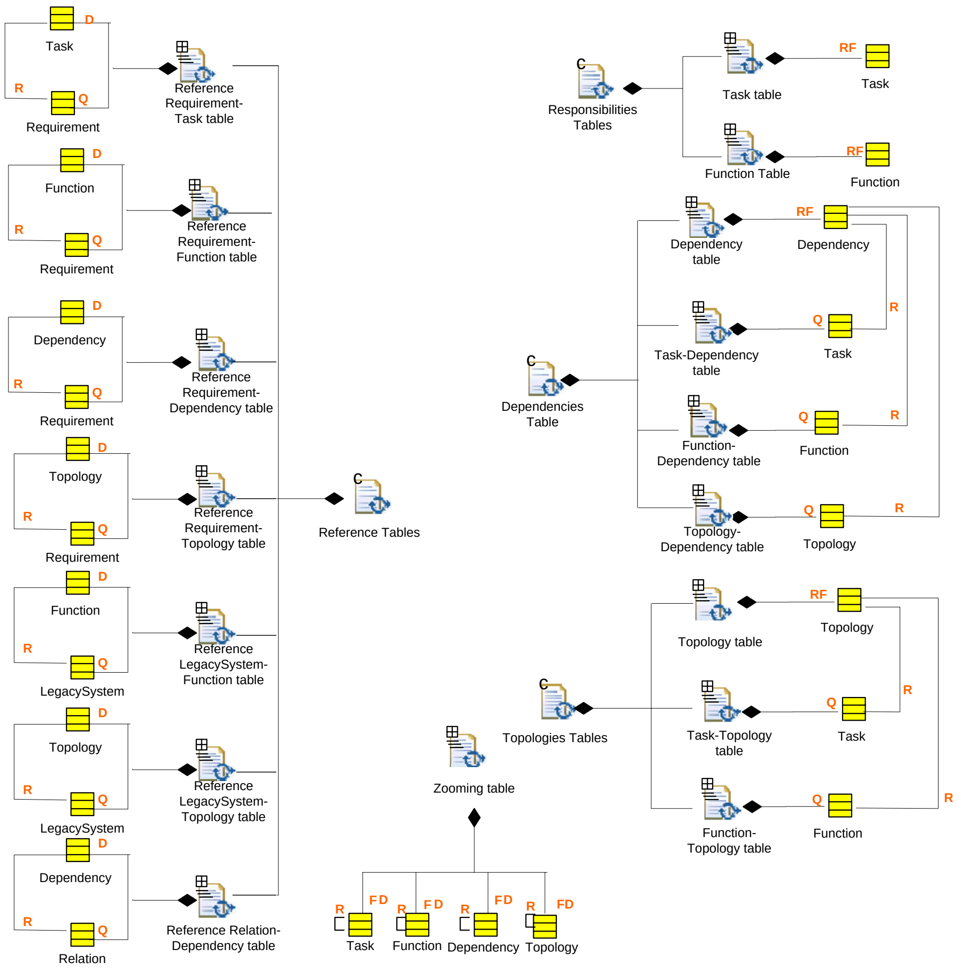

Figure 2.17: The Analysis work products

Requirement	Task
requirement name	*task names*

Requirement	Function
requirement name	*function names*

Requirement	Topology
requirement name	*topology names*

Requirement	Dependency
requirement name	*dependency names*

Legacy-System	Function
legacy-system name	*function names*

Legacy-System	Topology
legacy-system name	*topology names*

Relation	Dependency
relation name	*dependency names*

Figure 2.18: References Tables, in top-down order: $(L)RRT_t$, $(L)RRF_t$, $(L)RRTo_t$, $(L)RReqD_t$, $(L)RLSF_t$, $(L)RLST_t$, $(L)RRelD_t$.

this set of tables includes the Task table ($(L)T_t$), which lists all the tasks, and the Function table ($(L)F_t$), which lists all the functions.

Task	Description
task name	*task description*

Function	Description
function name	*function description*

Figure 2.19: Responsibilities Tables, in top-down order: $(L)T_t$, $(L)F_t$

Dependencies Tables (Figure 2.20) relate functions and tasks with each other. More precisely, the Dependency table ($(L)D_t$) describes all the dependencies among abstract entities, while the Task-Dependency table ($(L)TD_t$) specifies the set of dependencies where each task is involved, the Function-Dependency table ($(L)FD_t$) specifies the list of dependencies where each function is involved, and Topology-Dependency table ($(L)TopD_t$) specifies the list of dependencies where each topology is involved. Typically, when a requirement generates both a task and a function, the function is necessary to achieve the task. Correspondingly, other dependencies arise, in addition to the dependencies originating from the SODA relations.

Dependency	Description
dependency name	*dependency description*

Task	Dependency
task name	*dependency names*

Function	Dependency
function name	*dependency names*

Topology	Dependency
topology name	*dependency names*

Figure 2.20: Dependencies Tables in top-down order $(L)D_t$, $(L)TD_t$, $(L)FD_t$ and $(L)TopD_t$.

Topologies Tables (Figure 2.21), in turn, express the topological constraints over the environment. So, the Topology table ($(L)Top_t$) describes the topological constraints, while the Task-Topology table ($(L)TTop_t$) specifies the list of the topological constraints where each task is involved, and the Function-Topology table ($(L)FTop_t$) specifies the list of the topological constraints where each function is involved.

Topology	Description
Topology name	*topology description*

Task	Topology
task name	*topology names*

Function	Topology
function name	*topology names*

Figure 2.21: Topologies Tables in top-down order: $(L)Top_t$, $(L)TTop_t$, $(L)FTop_t$.

2.3 The Architectural Design

Figure 2.22 presents the Analysis process, while Figure 2.23 presents the flow of activities, the involved roles and the work products

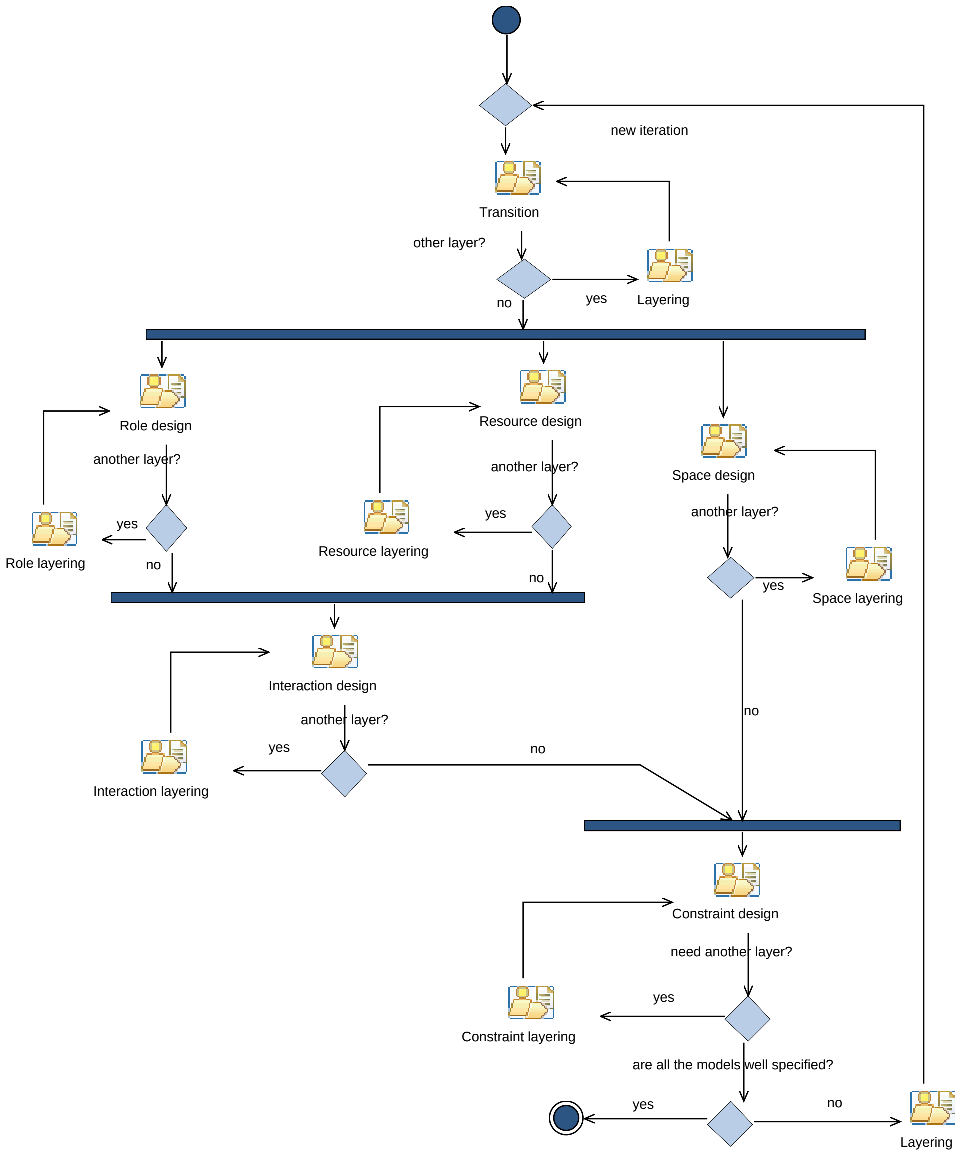

Figure 2.22: The Architectural Design process

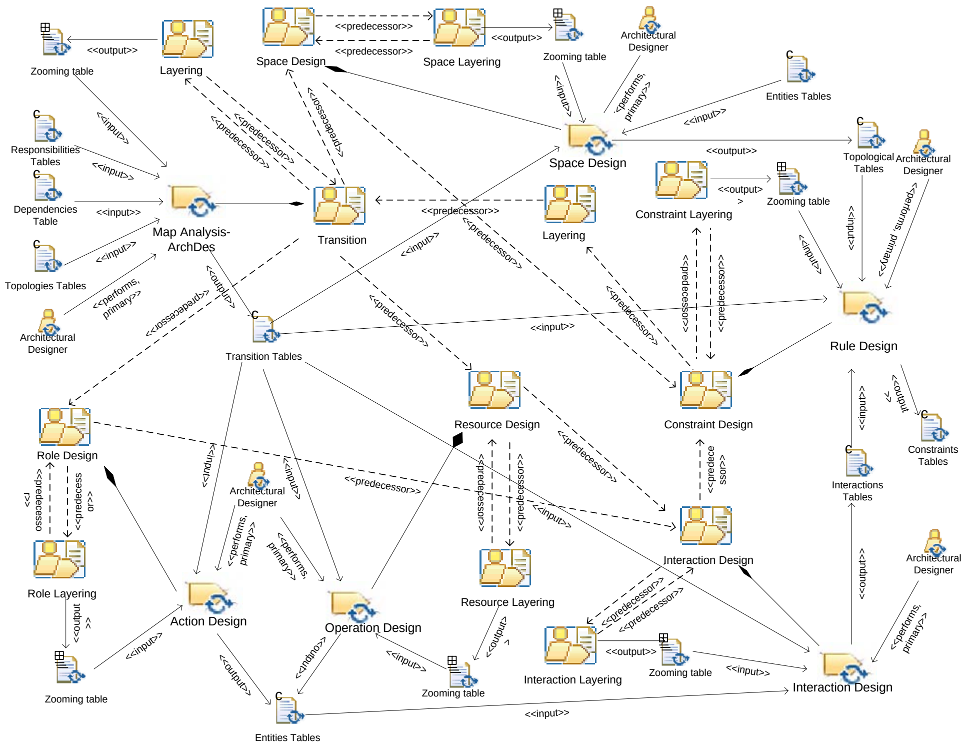

Figure 2.23: The Architectural Design flow of activities, roles and work products

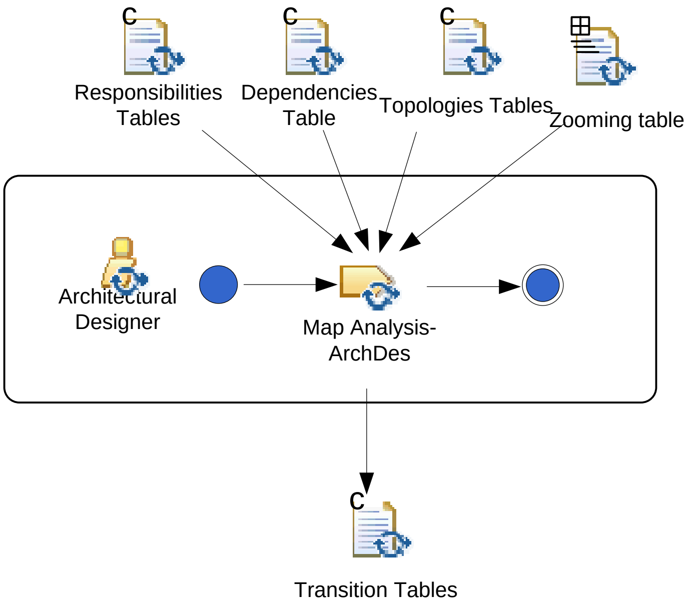

Figure 2.24: The flow of task in the Transition activity

2.3.1 Process roles

One role is involved in the Layering pattern: the Architectural Designer.

Architectural Designer

System Analyst is responsible for:

2.3.2 Activity Details

For the details about the different Layering activities please refer Section 1.3.

Transition activity

The flow of tasks inside this activity is reported in Figure 2.24 and the tasks are detailed in the following table.

Role Design activity

The flow of tasks inside this activity is reported in Figure 2.25 and the tasks are detailed in the following table.

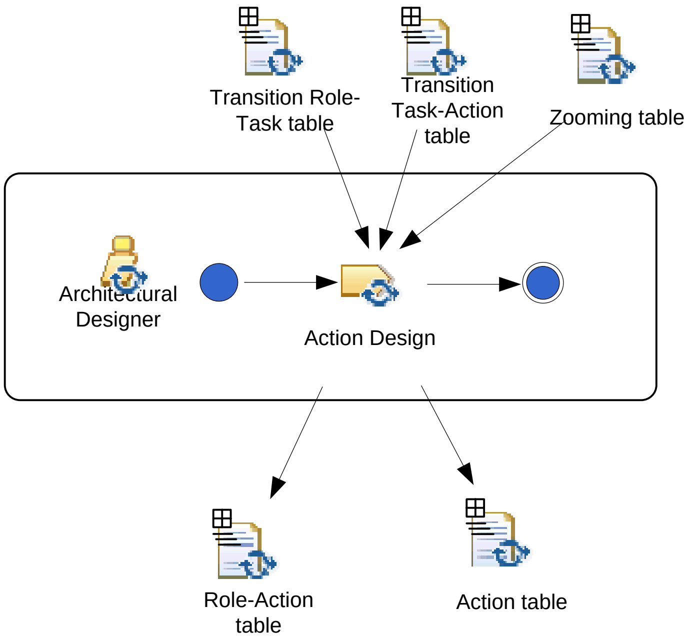

Figure 2.25: The flow of task in the Role Design activity

Resource Design activity

The flow of tasks inside this activity is reported in Figure 2.26 and the tasks are detailed in the following table.

Constraint Design activity

The flow of tasks inside this activity is reported in Figure 2.27 and the tasks are detailed in the following table.

Interaction Design activity

The flow of tasks inside this activity is reported in Figure 2.28 and the tasks are detailed in the following table.

Space Design activity

The flow of tasks inside this activity is reported in Figure 2.29 and the tasks are detailed in the following table.

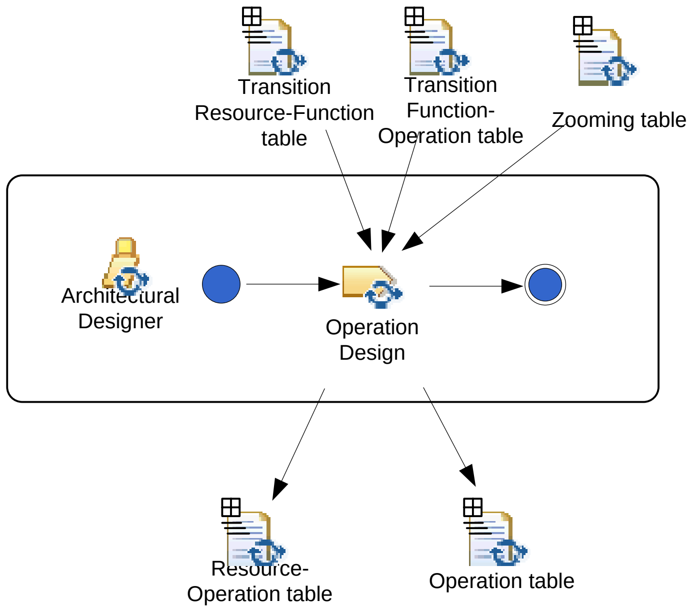

Figure 2.26: The flow of task in the Resource Design activity

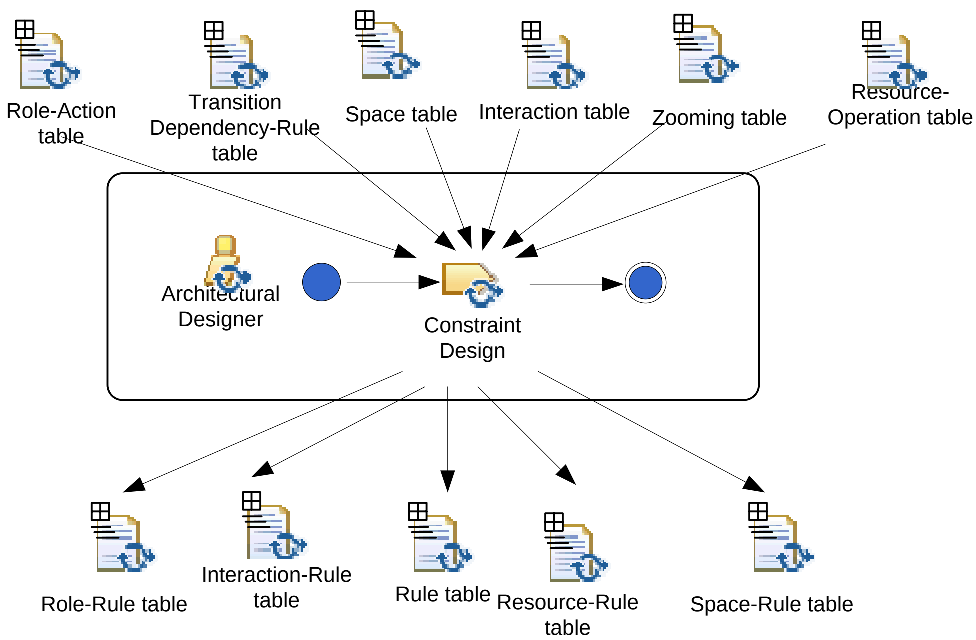

Figure 2.27: The flow of task in the Constraint Design activity

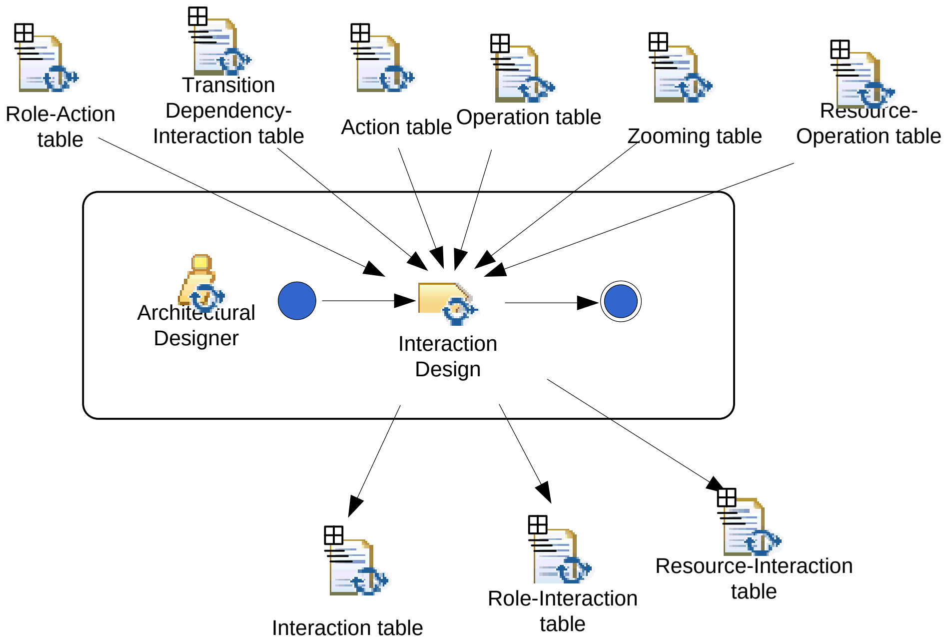

Figure 2.28: The flow of task in the Interaction Design activity

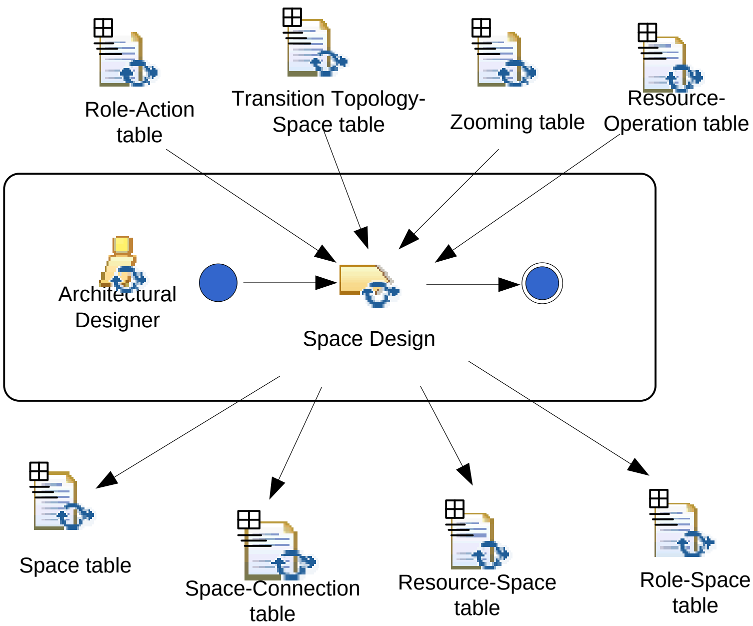

Figure 2.29: The flow of task in the Space Design activity

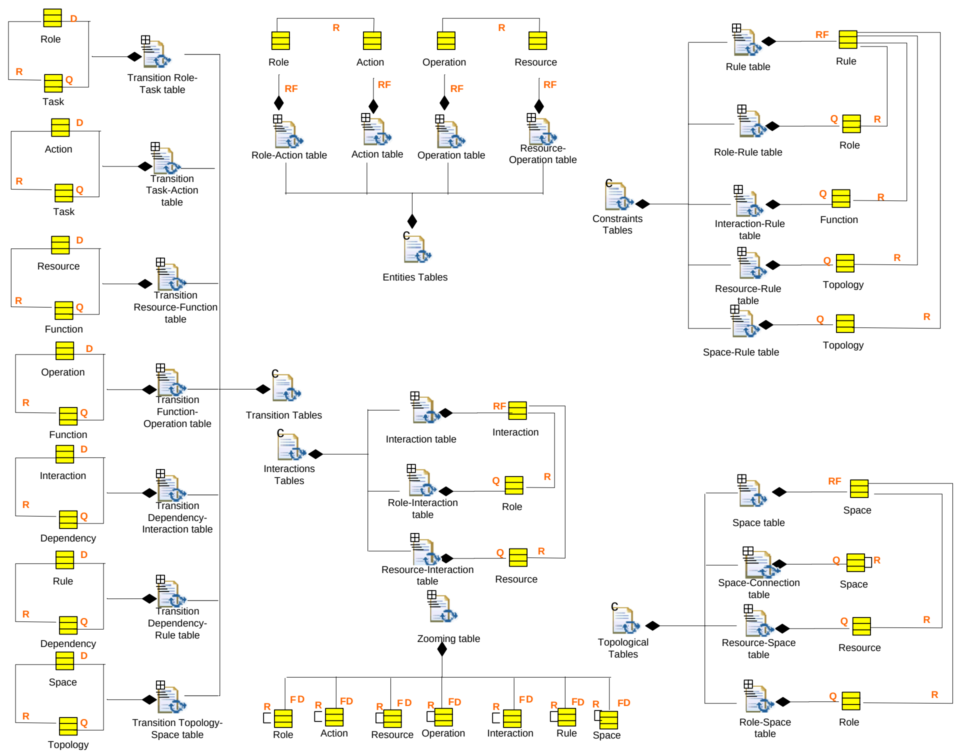

Figure 2.30: The Architectural Design work products

2.3.3 Work products

The Architectural Design step consists of four sets of tables: Transition Tables, Entities Tables, Interaction Tables and Topological Tables.

Their relationships with the MMMElements are described in Figure 2.30.

This diagram represents the Layering in terms of the Work Product and its relationships with the SODA meta-model (Section 1.2) elements.

Details of the tables

In order to link the Analysis step with the Architectural Design step, the Analysis entities are related to the Architectural Design by means of Transition Tables (Figure 2.31). So, for each layer, the Transition Role-Task table ($(L)TRT_t$) relates tasks and roles, the Transition Task-Action table ($(L)TTA_t$) relates tasks and actions, the Transition Resource-Function table ($(L)TRF_t$) links functions and resources, the Transition Function-Operation table ($(L)TFO_t$) links functions and operation, the Transition Interaction-Dependency table ($(L)TID_t$) maps dependencies onto interactions, the Transition Rule-Dependency table ($(L)TRuD_t$) maps dependencies onto rules, and the Transition Topology-Space table ($(L)TTopS_t$) specifies the mapping between topologies and spaces.

Role	Task
role name	*task names*

Task	Action
task name	*action names*

Resource	Function
resource name	*function names*

Function	Operation
function name	*operation names*

Dependency	Interaction
dependency name	*interaction names*

Dependency	Rule
dependency name	*rule names*

Topology	Space
topology name	*space names*

Figure 2.31: Transition Tables, in top-down order: $(L)TRT_t$, $(L)TTA_t$, $(L)TRF_t$, ($(L)TFO_t$), $(L)TID_t$, $(L)TTopS_t$.

Entities Tables (Figure 2.32) describe both the active entities (the roles) able to perform some action in the system, and the passive entities (the resources) which provide services. In particular, the Action table ($(L)A_t$) describes the actions executable by some roles, while the Operation table ($(L)O_t$) specifies the operations provided by resources. Then, the Role-Action table ($(L)RA_t$) specifies the actions that each role can do, while the Resource-Operation table ($(L)RO_t$) specifies the operations that each resource can provide.

Interactions Tables (Figure 2.33) describe the interaction between roles and resources: more precisely, the Interaction table ($(L)I_t$) defines the single interactions, the Action-Interaction table ($(L)AcI_t$) specifies the interactions where each action is involved, and the Operation-Interaction table ($(L)OpI_t$) specifies the interactions where each operation is involved.

Constraints Tables (Figure 2.34) describe the constraints over the entities behaviours: more precisely, the Rule table ($(L)Ru_t$) defines the single

Action	Description
action name	*description*

Operation	Description
operation name	*description*

Role	Action
role name	*action names*

Resource	Operation
resource name	*operation names*

Figure 2.32: Entities Tables, in top- down order: $(L)A_t$, $(L)O_t$, $(L)RA_t$, $(L)RO_t$

Interaction	Description
interaction name	*description*

Action	Interaction
action name	*interaction names*

Operation	Interaction
operation name	*interaction names*

Figure 2.33: Interactions Tables, in top- down order: $(L)I_t$, $(L)AcI_t$, $(L)OpI_t$

rule, the Rule-Interaction table ($(L)IRu_t$) specifies the constraints over the interactions, the Resource-Rule table ($(L)ReI_t$) specifies the rules where each resource is involved, the Role-Rule table ($(L)RoRu_t$) specifies the rules where each role is involved, and the Space-Rule table ($(L)SRu_t$) specifies the rules where each space is involved.

Finally, Topological Tables (Figure 2.35) describe the logical structure of the environment. More precisely, the Space table ($(L)S_t$) describes the spaces, the Space-Connection table ($(L)SC_t$) shows the connections among the spaces of a given layer (the hierarchical relations between spaces are expressed via the Zooming Table), and the Space-Resource table ($(L)SRe_t$) shows for each resources the spaces where it is involved. Similarly, the Space-Role table ($(L)sRo_t$) lists the spaces where each role is involved.

Rule	Description
rule name	*description*

Interaction	Rule
interaction name	*rule names*

Resource	Rule
resource name	*rule names*

Role	Rule
role name	*rule names*

Space	Rule
space name	*rule names*

Figure 2.34: Constraints Tables, in top- down order: $(L)Ru_t$, $(L)IRu_t$, $(L)ReRu_t$, $(L)RoRu_t$, $(L)SRu_t$

2.4 The Detailed Design

The goal of Detailed Design is to choose the most adequate representation level for each architectural entity, thus leading to depict one (detailed) design from the several potential alternatives outlined above. For the sake of concreteness, let us refer to Figure 2.36 (left), where the hypothesis that the Architectural Design phase outlined roles R1, R2 at the core layer C, roles R4, R5 and the projection of R2 at layer C+1, and roles R6, R7, R8 and R9 at layer C+2, is made. Turning this conceptual view into a real design view means to choose one representation (i.e., zoom) level for each entity, starting from the core layer: so, for instance, we could decide to zoom only R1, keeping R2 at the basic (core) representation level. Moreover, we could choose to further in-zoom R4 as the set of (sub)roles R6,R7, while keeping R5 as is. The result can be graphically expressed by *carving out* the roles R1, R2, R4, R5, R6 and R7 from the Architectural Design view, as shown with the curbed line in Figure 2.36 (centre).

A similar approach is adopted for the environmental entities: the un-zoomed resources identified in the previous step are now mapped onto suitable artifacts (intended as entities providing some services), and in-zoomed resources are mapped onto aggregates of artifacts.

This "carving operation" represents the boundary between Architectural Design – expressed in terms of roles, services, resources and workspaces – and Detailed Design—expressed in terms of agents, agent societies, artifacts and aggregates. In the case of our example (see Figure 2.36 (right)), role R1 is to

Space	Description
space name	*description*

Space	Connection
space name	*space names*

Resource	Space
resource name	*space names*

Role	Space
role name	*space names*

Figure 2.35: Topological Tables, in top-down order: $(L)S_t$, $(L)SC_t$, $(L)SRe_t$ and $(L)SRo_t$

be mapped onto an agent society (A1), while role R2 is to be mapped onto an individual agent (A2). Going further, the agent society A1 is composed of two entities, representing roles R4 and R5: again, the first is to be mapped onto an agent society (A4), since role R4 is in-zoomed in the carving, while R5 is to be mapped onto an individual agent (A5); the same process applies to A4, which turns out to be composed of agents A6 and A7, mapping roles R6 and R7, respectively.

Figure 2.37 presents the Analysis process, while Figure 2.38 presents the flow of activities, the involved roles and the work products

2.4.1 Process roles

One role is involved in the Layering pattern: the Detailed Designer.

Detailed Designer

System Analyst is responsible for:

2.4.2 Activity Details

Carving activity

The flow of tasks inside this activity is reported in Figure 2.39 and the tasks are detailed in the following table.

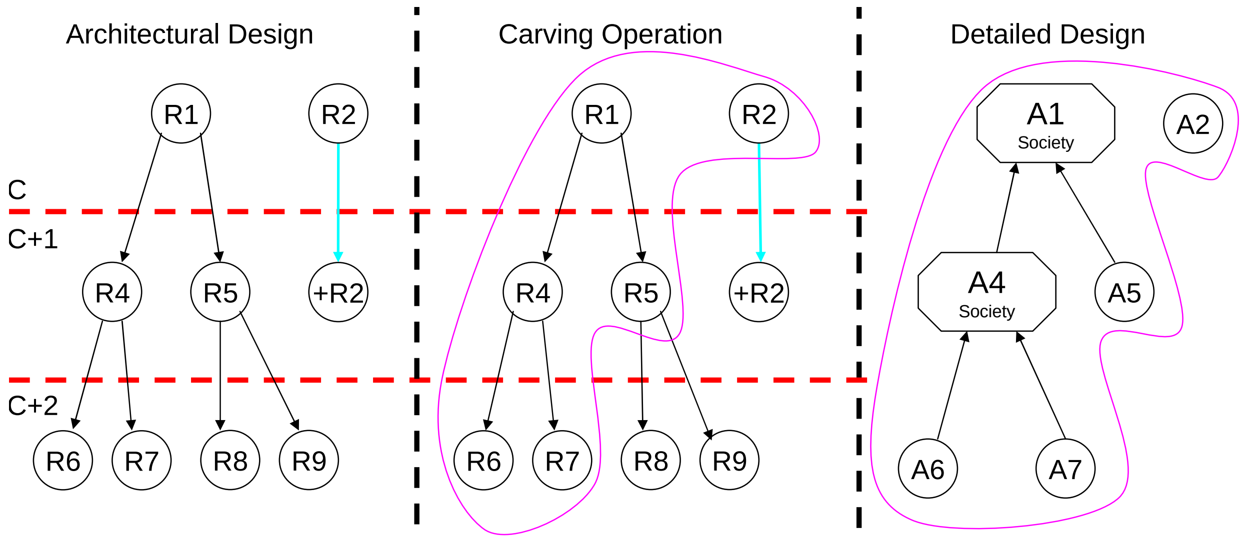

Figure 2.36: Design steps and Carving Operation

Mapping activity

The flow of tasks inside this activity is reported in Figure 2.40 and the tasks are detailed in the following table.

Agent Design activity

The flow of tasks inside this activity is reported in Figure 2.41 and the tasks are detailed in the following table.

Environment Design activity

The flow of tasks inside this activity is reported in Figure 2.42 and the tasks are detailed in the following table.

Interaction Detailed Design activity

The flow of tasks inside this activity is reported in Figure 2.43 and the tasks are detailed in the following table.

Workspace Design activity

The flow of tasks inside this activity is reported in Figure 2.44 and the tasks are detailed in the following table.

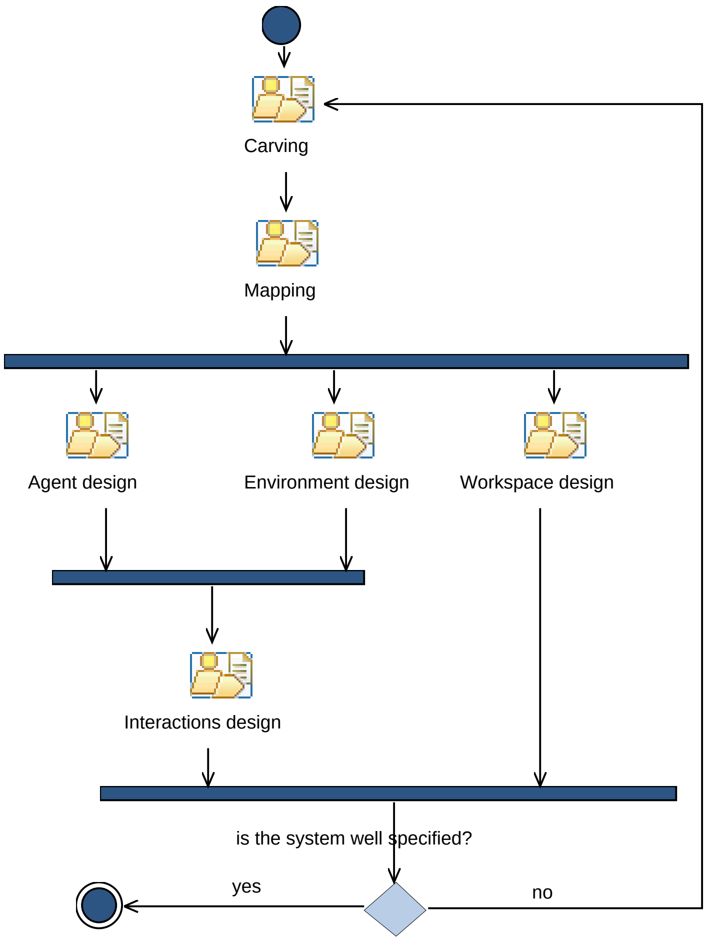

Figure 2.37: The Detailed Design process

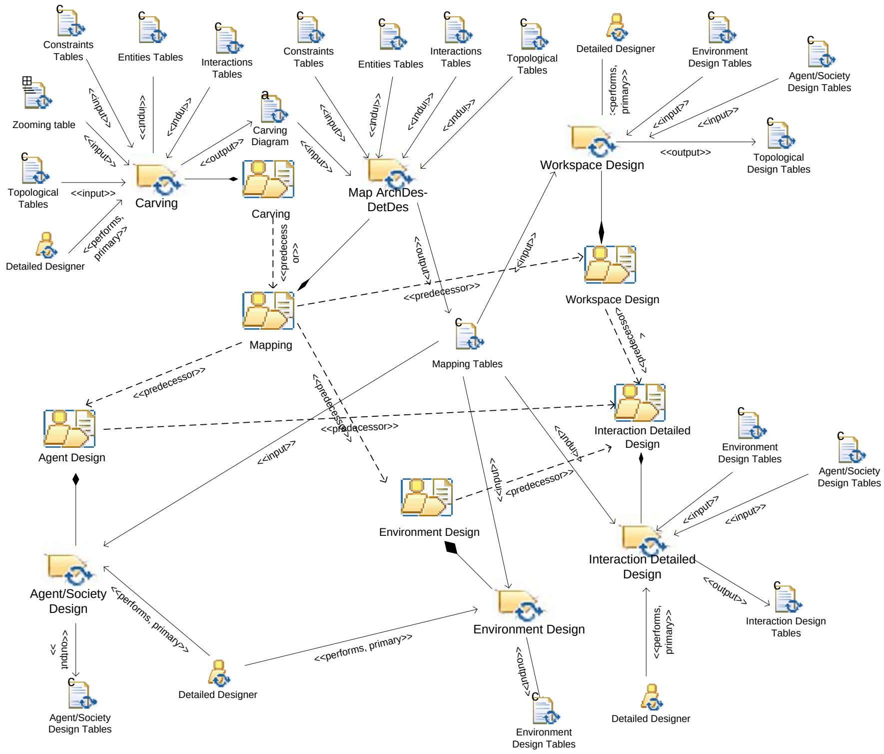

Figure 2.38: The Detailed Design flow of activities, roles and work products

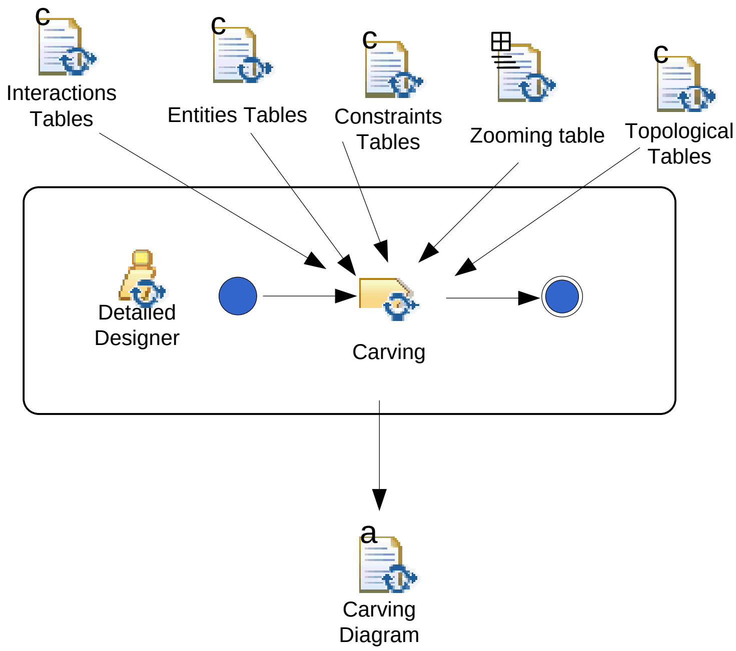

Figure 2.39: The flow of task in the Carving activity

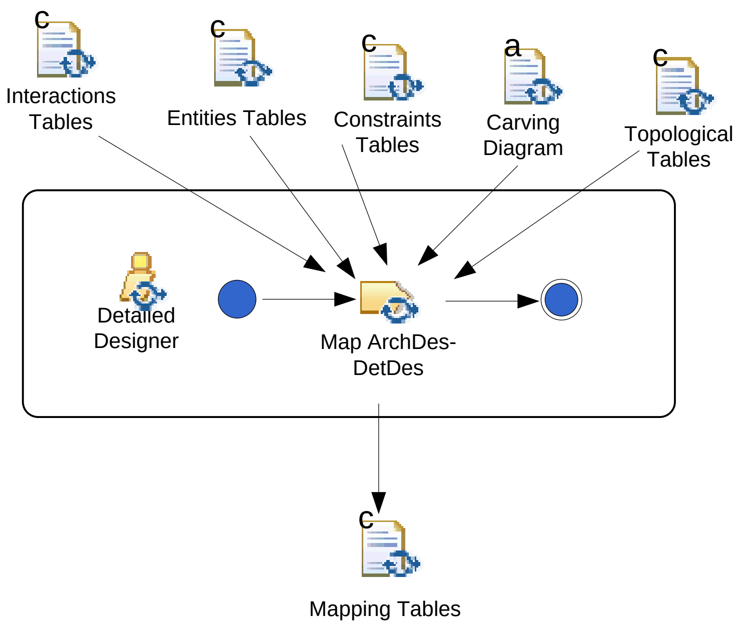

Figure 2.40: The flow of task in the Mapping activity

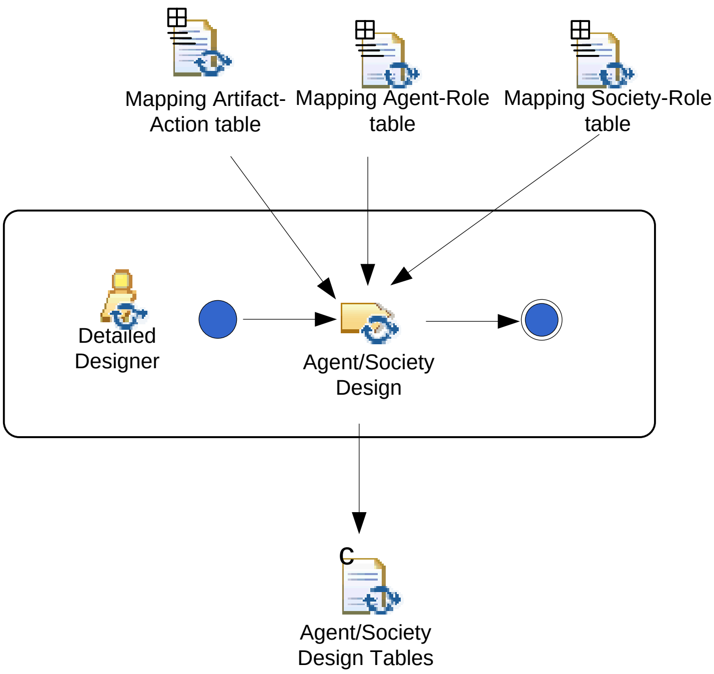

Figure 2.41: The flow of task in the Agent Design activity

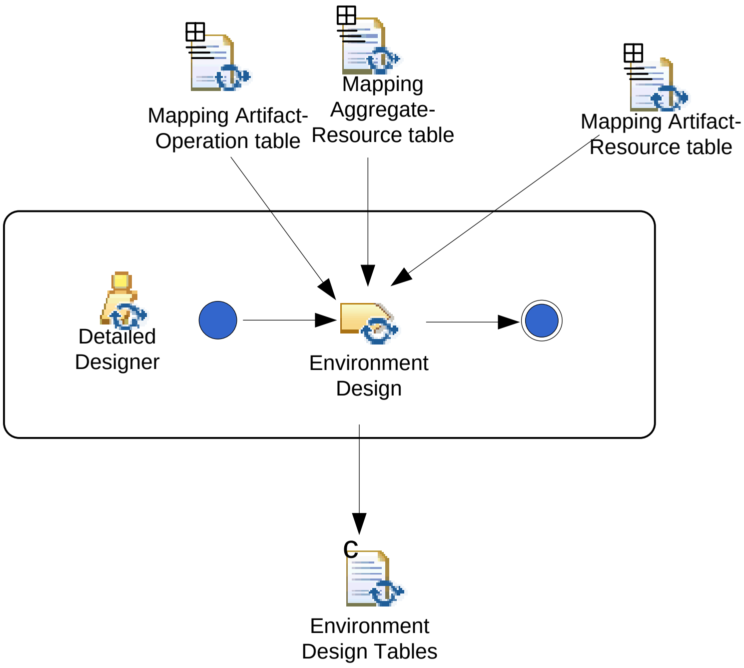

Figure 2.42: The flow of task in the Environment Design activity

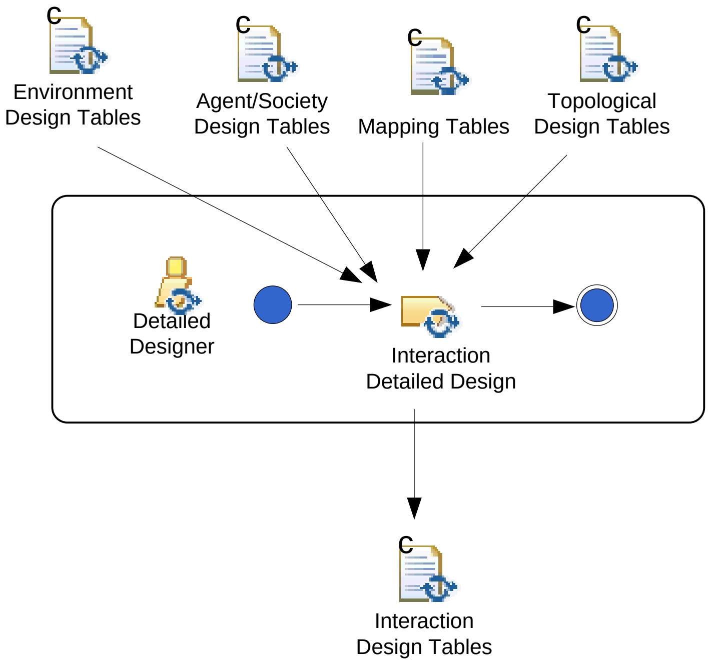

Figure 2.43: The flow of task in the Interaction Detailed Design activity

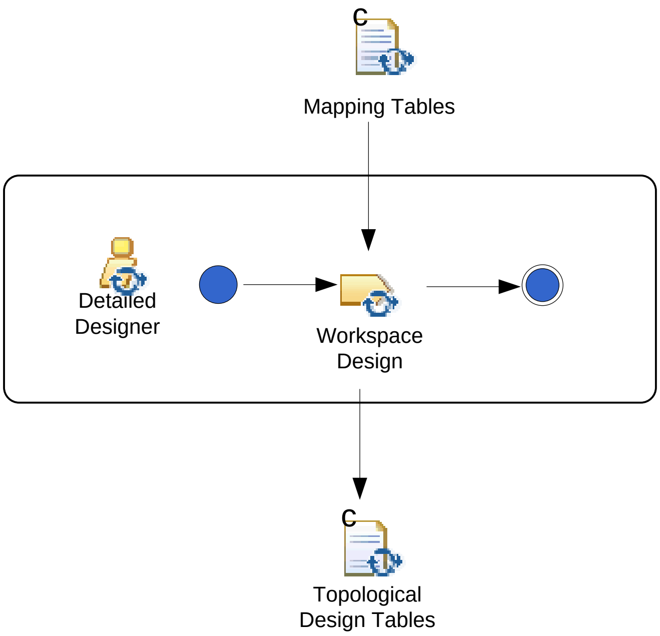

Figure 2.44: The flow of task in the Workspace Design activity

2.4.3 Work products

The Detailed Design step exploits several sets of tables: Mapping Tables, Agent/Society Design Tables, Environment Design Tables, Interaction Design Tables, and Topological Design Tables.

Their relationships with the MMMElements are described in Figure 2.45.

This diagram represents the Layering in terms of the Work Product and its relationships with the SODA meta-model (Section 1.2) elements.

Details of the tables

In order to link the Architectural Design step with the Detailed Design step, the Architectural Design entities are related to the Detailed Design by means of Mapping Tables (Figure 2.46). the Mapping Agent-Role table (MAR_t) maps roles onto agents, the Mapping Society-Role table (MSR_t) maps role onto society, the Mapping Artifact-Action table ($MAAc_t$) maps actions onto individual artifacts, the Mapping Artifact-Resource table ($MArR_t$) maps resources onto artifacts, the Mapping Aggregate-Resource table ($MAggR_t$) maps resources onto aggregate, the Mapping Artifact-Operation table ($MArOp_t$) maps operation onto environmental artifacts, the Mapping Artifact-Rule table ($MArRu_t$) maps the rules specified in the Architectural Design onto the

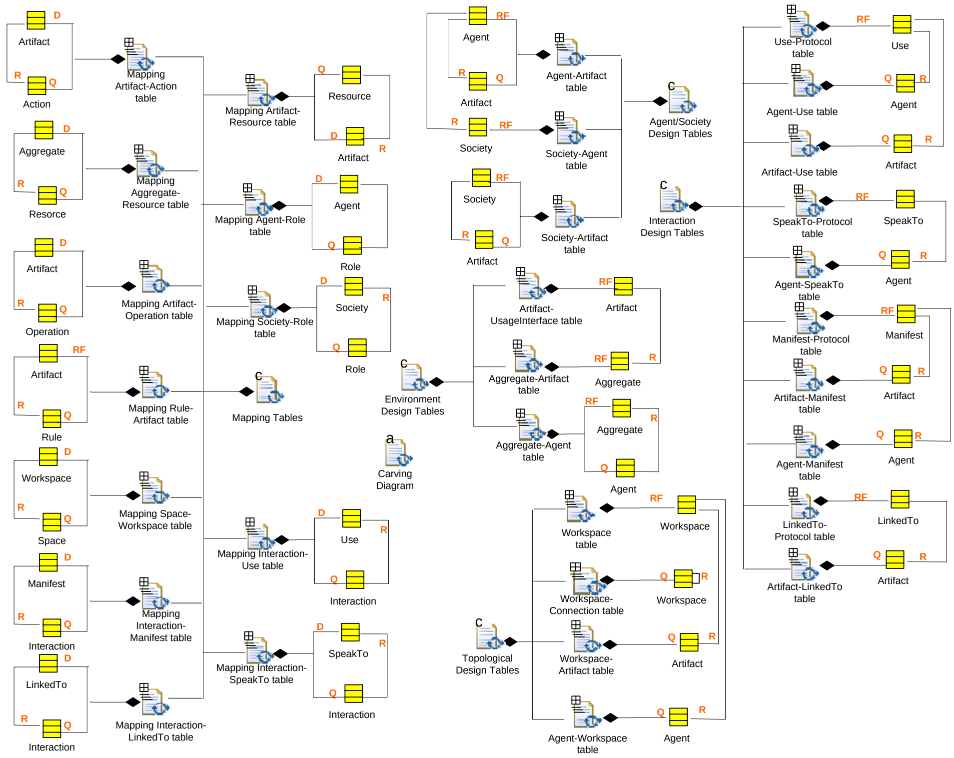

Figure 2.45: The Detailed Design work products

artifacts that implement and enforce them, the Mapping Artifact-Operation table (MSW_t) maps spaces onto workspaces, the Mapping Interaction-Use table (MIU_t) maps interactions onto uses, the Mapping Interaction-Manifest table (MIM_t) maps interactions onto manifests, the Mapping Interaction-SpeakTo table ($MISp_t$) maps interactions onto speak to, the Mapping Interaction-LinkedTo table (MIL_t) maps interactions onto linked to.

Agent/Society Design Tables (Figure 2.47) depicts agents, individual artifacts, and the agent societies derived from the carving operation. More precisely, the Agent-Artifact table (AA_t) specifies the individual artifacts related to each agent, the Society-Agent table (SA_t) lists the agents belonging to a specific society, and the Society-Artifact table (SAr_t) lists the artifacts belonging to a specific society.

In turn, Environment Design Tables concern the design of artifacts and workspaces: the Artifact-UsageInterface table (AUI_t) details the operations provided by each artifact, the Aggregate-Artifact table ($AggArt_t$) lists the artifacts belonging to a specific aggregate, while the Aggregate-Agent table ($AggAge_t$) lists the agents belonging to a specific aggregate Here the distinction between the artifact types is not presented because the design of the usage interface, the allocation of the artifacts to workspaces and the aggregates are done in the same way for all the artifact's types.

In turn, Interaction Design Tables (Figure 2.49)concern the design of interactions among entities: the Use-Protocol table (UP_t) details the protocols for each "use interaction", while the Agent-Use table ($AgeU_t$) and Artifact-Use table ($ArtU_t$) specify respectively the "use" where agent and artifacts are involved. The SpeakTo-Protocol table (SP_t) details the protocols for each "SpeakTo interaction", the Agent-SpeakTo table ($AgeSp_t$) specifies the "SpeakTo" where each agent is involved, the Manifest-Protocol table (MP_t) details the protocols for each "Manifest interaction", the Artifact-Manifest table ($ArtM_t$) and the Agent-Manifest table ($AgeM_t$) specify respectively the "Manifest" where each artifact and agent are involved. Finally, the LinkedTo-Protocol table (LP_t) details the protocols for each "LinkedTo" interaction, the Artifact-LinkedTo table ($ArtL_t$) specifies the "LinkedTo" where each artifact is involved.

Finally, Topological Design Tables (Figure 2.50) describe the structure of the environment. More precisely, the Workspace table ($(L)W_t$) describes the workspaces, the Workspace-Connection table ($(L)WC_t$) shows the connections among the workspaces, and the Workspace-Artifact table ($(L)WArt_t$) shows the allocation of the artifacts to workspaces. Similarly, the Workspace-Agent table ($(L)WA_t$) lists the workspaces that each agent can perceive.

Agent	Role
agent name	*role names*

Society	Role
society name	*role name*

(Individual) Artifact	Action
artifact name	*action names*

(Environmental) Artifact	Resource
artifact name	*resource names*

Aggregate	Resource
aggregate name	*resource name*

(Environmental) Artifact	Operation
artifact name	*operation names*

Rule	Artifact
rule name	*artifact names*

Workspace	Space
workspace name	*space names*

Interaction	Use
interaction name	*use names*

Interaction	Manifest
interaction name	*manifest names*

Interaction	SpeakTo
interaction name	*speak names*

Interaction	LinkedTo
interaction name	*linked names*

Figure 2.46: Mapping Tables in top- down order: MAR_t, MSR_t, $MAAc_t$, $MArR_t$, $MAggR_t$, $MArOp_t$, $MArRu_t$, MSW_t, MIU_t, MIM_t, $MISp_t$, MIL_t

Agent	(Individual) Artifact
agent name	*artifact names*

Society	Agent
Society name	*agent names*

Society	Artifact
society name	*artifact names*

Figure 2.47: Agent/Society Design Tables in top- down order: AA_t, SA_t, SAr_t

Artifact	Usage Interface
artifact name	*list of operations*

Aggregate	Artifact
aggregate name	*artifact names*

Aggregate	Agent
aggregate name	*agent names*

Figure 2.48: Environment Design Tables in top- down order: AUI_t, $AggArt_t$, $AggAge_t$

Use	Protocol
use name	*protocol description*

Agent	Use
agent name	*use names*

Artifact	Use
artifact name	*use names*

Speak To	Protocol
speak name	*protocol description*

Agent	SpeakTo
agent name	*speak names*

Manifest	Protocol
speak name	*protocol description*

Artifact	Manifest
artifact name	*manifest names*

Agent	Manifest
artifact name	*manifest names*

LinkedTo	Protocol
linked name	*protocol description*

Artifact	LinkedTo
artifact name	*linked names*

Figure 2.49: Interaction Design Tables in top-down order: UP_t, $AgeU_t$, $ArtU_t$, SP_t, $AgeSp_t$, MP_t, $ArtM_t$, $AgeM_t$, LP_t, $ArtL_t$

Workspace	Description
workspace name	*description*

Workspace	Connection
workspace name	*workspace names*

Workspace	Artifact
workspace name	*artifact names*

Agent	Workspace
agent name	*workspace names*

Figure 2.50: Topological Design Tables, in top-down order: $(L)W_t$, $(L)WC_t$, $(L)WArt_t$ and $(L)WA_t$

3 Work products dependencies

The diagram in Figure 3.1 describes the dependencies among the different composite work products.

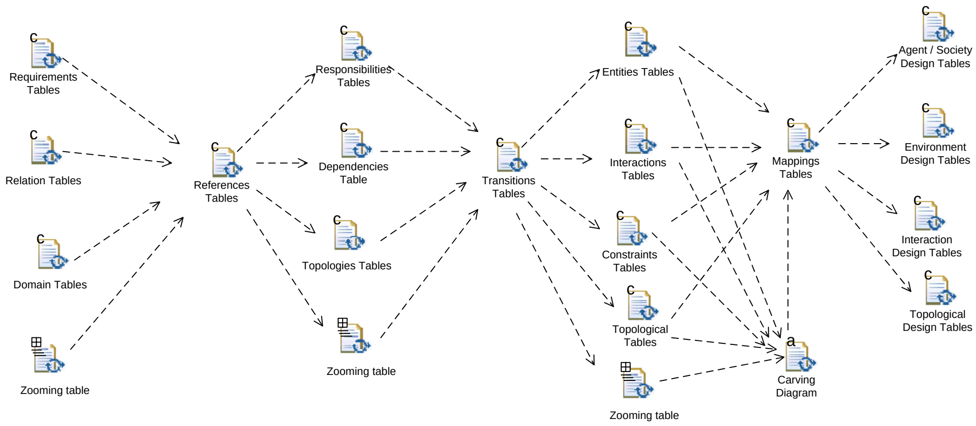

Figure 3.1: The work products dependencies

Bibliography

[1] aliCE Research Group. SODA home page. http://soda.alice.unibo.it.

[2] Ambra Molesini, Andrea Omicini, Enrico Denti, and Alessandro Ricci. SODA: A roadmap to artefacts. In Oğuz Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci, editors, *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer, June 2006. 6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised, Selected & Invited Papers.

[3] Ambra Molesini, Andrea Omicini, Alessandro Ricci, and Enrico Denti. Zooming multi-agent systems. In Jörg P. Müller and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *LNCS*, pages 81–93. Springer, 2006. 6th International Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.

[4] Object Management Group. Software & Systems Process Engineering Meta-Model Specification 2.0. `http://www.omg.org/spec/SPEM/2.0/PDF`, April 2008.

[5] Andrea Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer, 2001. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.

[6] Andrea Omicini. Formal ReSpecT in the A&A perspective. *Electronic Notes in Theoretical Computer Sciences*, 175(2):97–117, June 2007. 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.

[7] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artifacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 2005. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.