Towards an Explicit Bidirectional Requirement-to-Code Traceability Meta-model for the PASSI Methodology

Mihoub Mazouz[1], Farid Mokhati[2] and Mourad Badri[3]
[1]*Department of Mathematics and Computer Science, University of Oum El Bouaghi, Oum El Bouaghi, Algeria*
[2]*Department of Mathematics and Computer Science, University of Oum El Bouaghi, LAMIS Laboratory, Oum El Bouaghi, Algeria*
[3]*Department of Mathematics and Computer Science, Glog Laboratory, University of Quebec, Trois-Rivières, Canada*

Keywords: Multi-agent System, Traceability, PASSI.

Abstract: Traceability plays an important role in the development of computing systems, specifically, the complex ones. It provides several benefits to stakeholders and developers during the different phases of the systems development life cycle, including verification & validation and maintenance. Unfortunately, there are very few works in literature addressing the concept of traceability in multi-agent systems development methodologies. Having an incremental and iterative process, the well-known PASSI (Process for Agent Societies Specification and Implementation) methodology needs an explicit traceability in order to facilitate the understanding of the MAS under development and to better manage the changes occurring during the development process. In addition, it can lead to a requirement-based verification & validation. In this paper, we propose a new traceability meta-model for the PASSI methodology by introducing explicit traceability links of functional requirements through the various phases of the development life cycle.

1 INTRODUCTION

Tracing systems artifacts is a good factor to support various activities in the development process of computing systems (Spanoudakis, G, and A. Zisman., 2004). The introduction of traceability links between requirements and other systems artefacts can help the developer to better understand and manage the changes of any artifact on other artefacts that are connected thereto. It also gives a better development process visibility, as it facilitates the maintenance and verification & validation activities (Kannenberg et al., 2009). Integrating traceability in the development processes increases the quality of developed systems (Spanoudakis, G, and A. Zisman., 2004). The agent paradigm has demonstrated its effectiveness in modeling and design of complex systems. Several methodologies are now available in the literature to guide developers in the development of agent-oriented systems (e.g., PASSI (Cossentino, M., 2005, Cossentino, M. and V. Seidita, 2014), Gaia (Cernuzzi, L., et al., 2004), Prometheus (Winikoff, M. and L. Padgham, 2004), Tropos (Giorgini, P., et al., 2004).

Castro et al. reported in (Andréa C., et al., 2002) that the success in the development of the next generation of agent-oriented systems would be made thanks to the adoption of requirements traceability. Unfortunately, few works (Pinto, R., et al., 2007, Cysneiros G. and A. Zisman., 2007a, Cysneiros G. and A. Zisman., 2007b, Andréa C., et al., 2002, Castro, J., et al., 2003, Cysneiros, G. and A. Zisman, 2008, Pinto, R, et al., 2005) being adopted the traceability concept for multi-agent systems (MAS) development methodologies and associated tools exist. Introducing explicit traceability links between the different artifacts composing a system during the different phases of development facilitates the development process.

In this paper, we propose a new traceability meta-model for the well-known PASSI methodology (Cossentino, M., 2005). Traceability information are added to the specifications of the various system artifacts produced during the development process, allowing a bidirectional requirement-to-code traceability from the identification of the functional requirements in the domain requirement description phase to the deployment of agents that will achieve these requirements in the last phase of PASSI

Mazouz M., Mokhati F. and Badri M..
Towards an Explicit Bidirectional Requirement-to-Code Traceability Meta-model for the PASSI Methodology.
DOI: 10.5220/0005249102030209
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 203-209
ISBN: 978-989-758-073-4

process, deployment configuration.

The remainder of this paper is organized as follows. In Section 2, we give an overview of major related work. In Section 3, we give a brief description of the PASSI methodology. We introduce, in Section 4, the proposed traceability meta-model for the PASSI methodology. In Section 5, a case study is showed. Finally, Section 6 gives some conclusions and future work directions.

2 RELATED WORK

Over the last years, very few works (Pinto, R., et al., 2007, Cysneiros G. and A. Zisman., 2007a, Cysneiros G. and A. Zisman., 2007b, Andréa C., et al., 2002, Castro, J., et al., 2003, Cysneiros, G. and A. Zisman, 2008, Pinto, R, et al., 2005) addressing traceability in MAS development methodologies have emerged in literature. G. Cysneiros et al. (Cysneiros G. and A. Zisman., 2007a, Cysneiros G. and A. Zisman., 2007b, Cysneiros, G. and A. Zisman, 2008) address the Prometheus methodology. The authors proposed a rule-based approach supporting traceability and verification of completeness of artefacts represented in the Prometheus design models and JACK code specification. They have identified some types of traceability relationships between Prometheus design models artefacts and JACK code. The proposed rules allow the generation of traceability relationships after the models are built in an automatic way, which may neglect many relationships.

The other works (Pinto, R., et al., 2007, Andréa C., et al., 2002, Castro, J., et al., 2003, Pinto, R, et al., 2005) address the Tropos methodology. The authors have proposed an agent-oriented traceability reference model and showed how it can be used in the context of Tropos.

We present, in this paper, a new and explicit bidirectional Requirement-to-Code traceability for the PASSI methodology. Compared to the other approaches quoted above, the creation of traceability links we propose is performed when building PASSI models, covering the whole PASSI Process.

3 THE PASSI METHODOLOY

3.1 Description

PASSI (Process for Agent Societies Specification and Implementation) (Cossentino, M., 2005, Cossentino, M. and V. Seidita, 2014), is a step-by-step requirement-to-code methodology for designing

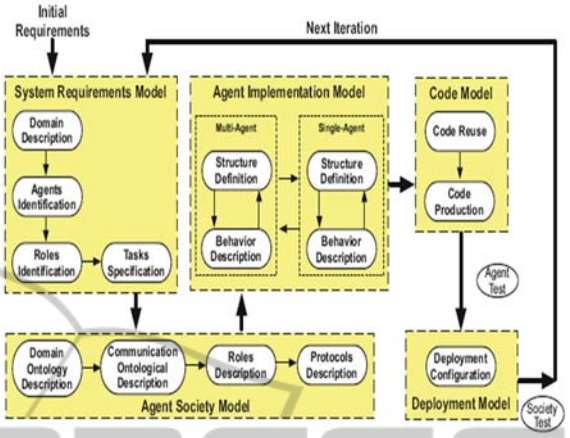

Figure 1: The PASSI design process. (Cossentino, M., 2005).

and developing agent-oriented systems. The process model of PASSI is iterative/incremental and includes five models, each model includes one or more phases (see Figure 1). The five models of PASSI are:

- **System Requirements Model:** It is composed of four phases: 1) *Domain Requirements Description*, as a use case diagram, the functional requirements of the system are described. 2) *Agents Identification*, where the agents making up the system to be developed are identified as packages including their proper functionalities. 3) *Roles Identification*, where the roles played by agents in the different scenarios of the system are identified in a series of sequence diagrams exploring the use cases identified in the first phase. 4) *Tasks specification*, where an activity diagram is used to specify the plan of each agent.
- **Agent Society Model:** It is composed of four phases: 1) *Domain Ontology Description*, where the agent knowledge is described by an ontology (in terms of Concept, Predicate, Action) and represented as a class diagram. 2) *Communication Ontological description*, where a class diagram is used to represent all agents, all interactions between them with the precision of the semantics of each communication between agents (ontology element, content language and the interaction protocol). 3) *Roles Description*, where roles already identified in the Role Identification phase for each agent (a package here) are represented by classes, each one is responsible for its own tasks (class operations). 4) *Protocols Description*,

AUML sequence diagrams are used to describe all non-standard protocols.

- **Agent Implementation Model:** It is composed of two phases: 1) *Agent Structure Definition*, class diagrams are used to describe the system architecture, 2) *Agent Behaviour Description*, where activity diagrams or state-machines can be used to describe the system behaviour. These two phases are views of two different abstraction levels: a) *society*, i.e. multi-agent abstraction level; b) *single agent* abstraction level.
- **Code Model:** It is composed of two phases: 1) *Code-Reuse*, in this phase, design patterns already exist can be used directly, 2) *Code Production*, where the skeleton of the source code of the system is automatically generated by the PTK (a Rational Rose plug-in that offers a support for PASSI) and a manual completion of the generated code is then achieved by the developer.
- **Deployment Model:** It is composed of one phase: *Deployment Configuration*, where deployment diagram is used to describe the allocation of agents to different processing units and any constraints on agent migration and mobility.

As illustrated by Figure 1, the PASSI process includes a test activity divided into two different levels: 1) *single-agent test*: when a framework built on top of JADE is implemented (Caire, G, et al., 2004). The most framework classes are "Test" class for testing a specific task of an agent and "*TestGroup*" class for testing all tasks composing a specific agent. 2) *society test:* at this level, integration verification is carried out together with the validation of the overall results of the current iteration (Cossentino, M., 2005).

3.2 The MAS Meta-Model of PASSI

The meta-model adopted for PASSI MAS is divided into three areas (Cossentino, M. and V. Seidita, 2014):

- **Problem domain:** where the elements describing the requirements that will be achieved by the future system are included. These elements are directly connected to the System Requirements Model.
- **Agency domain:** where the elements describing the multi-agent society in terms of environment (defined by a set of ontological elements) and the social aspect of agents (interaction between them) are included. The items of this area are connected directly to the Agent Society Model.
- **Solution domain:** where the elements describing the architectural solution (respecting the architecture of FIPA) of the problem in terms of agent classes, task class, agent code and task code are included. The elements of this area are connected directly to the two models: Agent Implementation and Code.

3.3 Traceability in PASSI

In (Cossentino, M. and V. Seidita, 2014), the authors have mentioned dependencies between the different artefacts produced during the PASSI process and how the diagrams are constructed from the preceding ones. These dependencies represent implicit links between artefacts, i.e. links which exist between artefacts having the same name, for example, between *Task* and *Agency_Task*, between Agent and Agency_Agent. However, no explicit traceability links were adopted to follow functional requirements since their identification in the Domain Requirements Description phase until the deployment of agents carrying out these requirements in Deployment Configuration phase. The lack of these traceability links causes some difficulties in the development of MAS using PTK. In fact, if the developer decides, using PTK, to change the name of task (Agency_Task) the corresponding task (*Task*) in Tasks Specification phase remains with the first name, which needs another modification. This takes more time. In the next section, we propose an explicit traceability links for the PASSI methodology, where it will be possible to follow the requirements during the whole PASSI development process. This will facilitate changes management.

4 EXPLICIT TRACEABILITY FOR PASSI

We introduce, in this section, a new traceability meta-model (Figure 2) for supporting explicit bidirectional Requirement-to-Code Traceability for the PASSI methodology. To create traceability links between the different artefacts, some information has to be added to the specifications of these artefacts in order to ensure traceability in both backward & forward directions: 1) a unique identifier "id" (ID according to the XML DTD syntax); 2) a name, and 3) a reference (s) (IDREF /

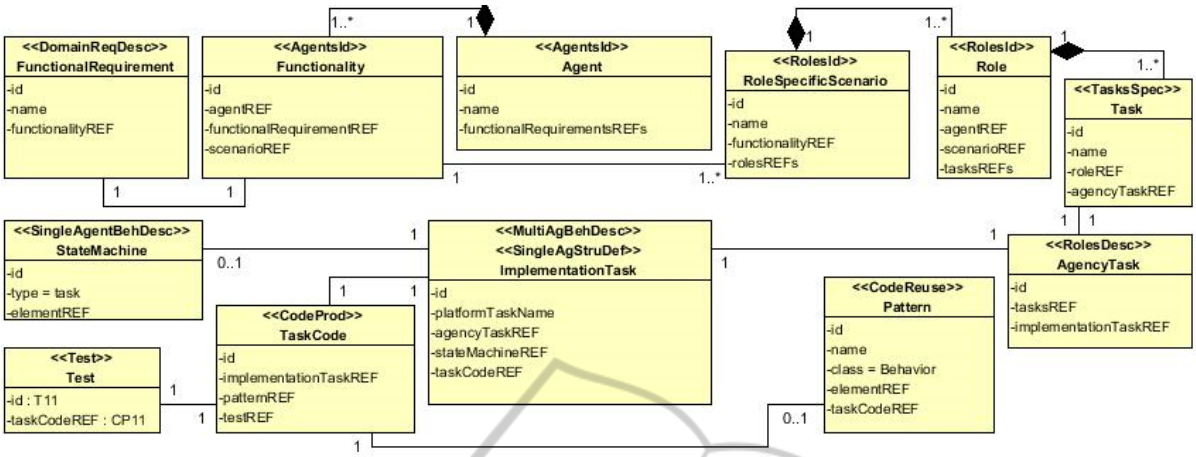

Figure 2: Requirement-to-Code Traceability Meta-model for PASSI.

Table 1: Functional requirement & Agent Life cycle.

System Requirements Model				Agent Society Model				Agent Implementation Model				Code Model		Test	Deployment Model
D.R.D.	A.Id.	R.Id.	T.Sp.	D.O.D.	Co.O.D.	R.D.	P.D.	M.A.S.D.	M.A.B.D.	S.A.S.D.	S.A.B.D	C.R.	C.P.	AgentTest	D.C.
FunctionalRequirement	Functionnality	RoleSpecificScenario_i	Task_j	/	/	Agency_Task_j	/	Implementation_Task_j	Implementation_Task_j	Implementation_Task_j	State Machine_j:1..k	Pattern_j	Task_Code_j	Test_j	/
	Agent	Role 1..*	Plan	/	Agency_Agent	Agency_Agent	/	Implementation_Agent	Implementation_Task 1..*	Implementation Agent	StateMachine	Pattern	Agent_Code	TestGroup	Node

IDREFS according to the XML DTD syntax) towards the precedent element(s) and / or towards the following element(s). Each *functional requirement* identified in the domain requirement description is assigned to a specific agent as a *functionality* (*agentREF* attribute) in the agents identification phase. One or more *RoleSpecificScenario* (composed of one or more "*Role*") explore(s) each *Functionality* (use case) in the Roles Identification phase. From each *Role*, several *Tasks* are identified in the tasks specification phase and so on until the Test phase. By creating traceability links, the developer can know, at any given moment, the origin of a *Task class* (the functional requirement from which is identified) in the single agent structure definition phase, and what is the Test according to it.

The creation of traceability links is supposed to be performed by the developer thanks to a new tool (under development) that can replace the official PTK by adding support for traceability introduced for PASSI with other improvements.

After identifying functional requirements in the first phase and during the Roles Identification phase, the developer will have to associate each functional requirement to a specific scenario (RoleSpecificScenario) thanks to a GUI and the links which will be created in both forward and backward directions.

The developer will need during each phase to

relate system artefacts.

Table 1 shows the life cycle of two PASSI elements: *Functional Requirement and Agent*. These two elements (and others) have to be followed up to the PASSI process end. For example, each *"agent"* (identified in *Agent Identification phase*) has to be linked to a "*plan*" (described in *Tasks Specification phase),* and so on (*Agency_Agent, Implementation_Agent,, Implementation_Tasks, StateMachine, Pattern, Agent_Code, TestGroup*) until the *"Node"* element (identified in *deployment configuration phase*). All these artefacts have to be linked one to each other in the two directions.

5 CASE STUDY

In order to illustrate the use of the meta-model we propose, we have selected a case study based on the "Juul Møller Bokhandel A/S" problem designed in (Cossentino, M., 2005) as an agent-oriented system using the PASSI methodology. Among the functional requirements that are presented in the Domain Requirements Description diagram, the « Provide Books » one (Figure 3).

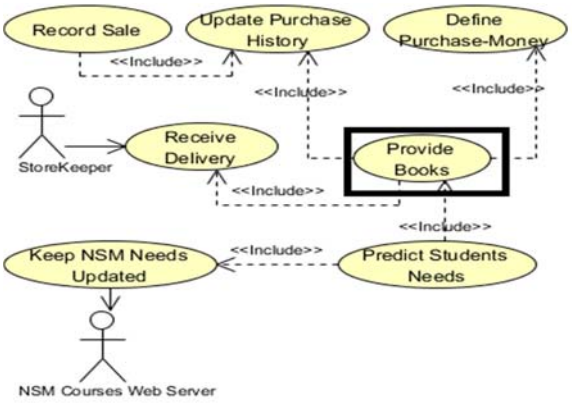

Figure 3: A portion of the domain requirements description diagram (Cossentino, M., 2005).

The functional requirement "Provide books" becomes a functionality assigned to the agent "PurchaserManager" in the agents identification phase (figure 4).

The scenario "Announcement of the need of a book purchase" (Figure 5) then explores the functionality mentioned above. Several agent roles participate in this scenario: PurchaseMonitor, PurchaseManager, PurchaseAdvisor, Purchaser and StoreUI. From each agent role, several tasks are identified like "ReceivePurchaseRequest" and "AskForAdvice" for the "BooksProvider" role (Figure 6).

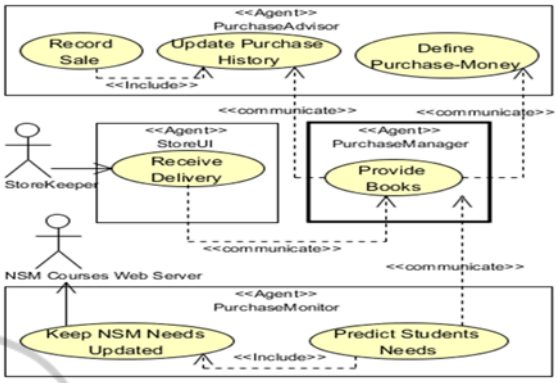

Figure 4: A portion of the agents' identification diagram (framed in bold, the functionality is assigned to the PurchaseManager Agent) (Cossentino, M., 2005).

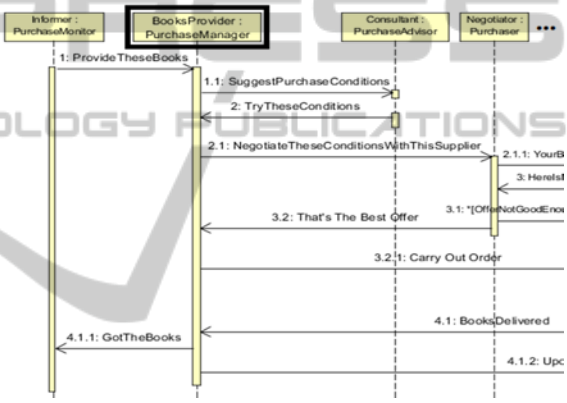

Figure 5: Roles identification diagram "Announcement of the need of a book purchase" scenario (framed in bold, the BooksProvider agent role) (Cossentino, M., 2005).

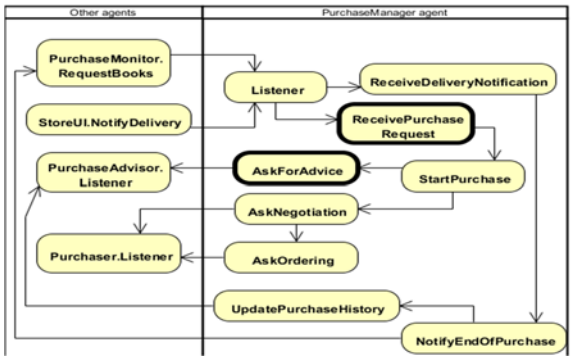

Figure 6: Tasks specification diagram for the agent "Purchase Manager" (framed in bold, the two tasks related to the "BooksProvider" role) (Cossentino, M., 2005).

Figure 7 shows a part of the traceability model (instance of the meta-model we proposed) for the functional requirement "providebooks". naturally,

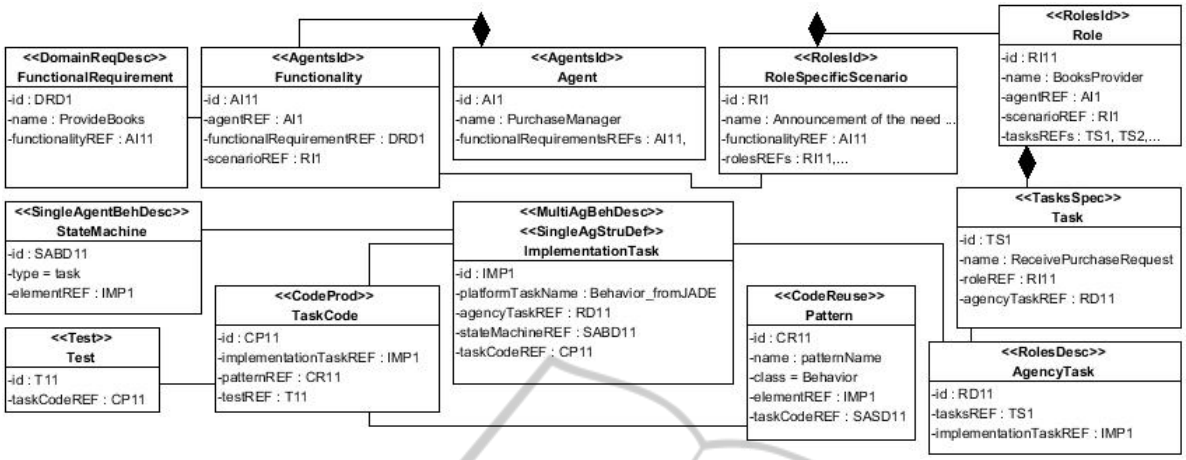

Figure 7: A part of «Provide Books» Functional Requirement-To-Code traceability model.

all the concepts that appear in the obtained traceability model (figure 6) are included in the traceability meta-model we have proposed.

6 CONCLUSION AND FUTURE WORK

Traceability is an important activity in the development of high quality computing systems. In this paper, we presented our attempt to introduce an explicit traceability links between artefacts produced during all the PASSI process. The existence of these links adds the ability to follow system's artefacts, particularly, the functional requirements during all the PASSI development phases. This will allow a better management of changes. As future work, we plan to: improve the traceability model proposed in this paper and develop a supporting tool with other additions to PTK.

REFERENCES

Cossentino, M. (2005). From Requirements to Code with the PASSI Methodology. In Agent-Oriented Methodologies, B. Henderson-Sellers and P. Giorgini (Editors). Idea Group Inc., Hershey, PA, USA.

Pinto, R., et al. (2007). A Traceability Reference Model for Agent Oriented Development. Proceedings of the Third Workshop on Software Engineering for Agent oriented Systems. João Pessoa. pp. 27-38. .

Cysneiros G. and A. Zisman. (2007a). Tracing AgentOriented Systems. In Proc. of the Grand Challenge Traceability Symposium, USA.

Cysneiros G. and A. Zisman. (2007b). Traceability for Agent-Oriented Design Models and Code, 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007), USA.

Andréa C., et al. (2002). Towards Requirement Traceability in TROPOS. WER 2004: 189-200.

Castro, J., et al. (2003). Requirements Traceability in Agent Oriented Development. Software Engineering for Large-Scale Multi-Agent Systems. A. Garcia, C. Lucena, F. Zambonelli, A. Omicini and J. Castro, Springer Berlin Heidelberg. 2603: 57-72.

Cossentino, M. and V. Seidita (2014). PASSI: Process for Agent Societies Specification and Implementation. Handbook on Agent-Oriented Design Processes. M. Cossentino, V. Hilaire, A. Molesini and V. Seidita, Springer Berlin Heidelberg: 287-329.

Cysneiros, G. and A. Zisman (2008). Traceability and completeness checking for agent-oriented systems. Proceedings of the 2008 ACM symposium on Applied computing. Fortaleza, Ceara, Brazil, ACM: 71-77.

Kannenberg et al. (2009). Why Software Requirements Traceability Remains a Challenge. CrossTalk: The Journal of Defense Software Engineering.

Spanoudakis, G, and A. Zisman. (2004). Software traceability: A roadmap, Handbook of Software Engineering and Knowledge Engineering, vol. 3, pp. 395–428.

Pinto, R, et al. (2005). Support for Requirement Traceability: The Tropos Case. 19th Simpósio Brasileiro de Engenharia de Software (SBES'05). Brasil.

Cernuzzi, L., et al. (2004). The Gaia Methodology. Methodologies and Software Engineering for Agent Systems. F. Bergenti, M.-P. Gleizes and F. Zambonelli, Springer US. 11: 69-88.

Winikoff, M. and L. Padgham (2004). The Prometheus Methodology. Methodologies and Software Engineering for Agent Systems. F. Bergenti, M.-P. Gleizes and F. Zambonelli, Springer US. 11: 217-234.

Giorgini, P., et al. (2004). The Tropos Methodology. Methodologies and Software Engineering for Agent Systems. F. Bergenti, M.-P. Gleizes and F. Zambonelli, Springer US. 11: 89-106.

Caire, G, et al. (2004). *Multi-agent systems implementation and testing*. In Fourth International Symposium: From Agent Theory to Agent Implementation.