Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Tschau Sepp
LOGIC Sub-Component
Software Requirements Specification

- **Authors:**
  Alexandru Dima [1]
  Olivier Clerc [2]
  Alejandro García [3]

- **Document number:**
  TS-LOGIC-SRS-001
- **Total number of pages:**
  30
- **Date:**
  Tuesday 3rd November, 2009
- **Location:**
  Zürich, Switzerland

[1] *E-mail: adima@student.ethz.ch*
[2] *E-mail: clerco@student.ethz.ch*
[3] *E-mail: garciaa@student.ethz.ch*

Contents

Revision & Sign-off Sheet

Date	Author	Version	Change Reference
2009-10-17	Alejandro García	0.1	Adding structure
2009-10-18	Olivier Clerc	0.2	Writing first draft
2009-10-18	Alexandru Dima	0.3	Writing first draft
2009-10-26	Olivier Clerc	0.4	Review and Rewriting
2009-10-26	Alexandru Dima	0.5	Adding requirements
2009-10-26	Alejandro García	0.6	Rewriting Overall description
2009-10-27	Alexandru Dima	1.0	Final version
2009-11-03	Alejandro García	1.1	Appendix

1 Introduction

1.1 Purpose

This document represents the Software Requirements Specification (SRS) for the `LOGIC` sub-component of the *Tschau Sepp Game Component*. It is designed and written for the stake holders, such as the teaching assistants, professors and developers involved in the project. Its purpose is to describe the scope, both the functional and non-functional software requirements, as well as the design constraints of the whole `LOGIC` sub-component. Furthermore, this document shows how the system's interfaces are designed in detail.

1.2 Scope

The *Tschau Sepp Game Component* is an implementation of the Swiss card game Tschau Sepp to be used by the overall *Multiplayer Card Games* system. For a better description of the scope of the system, the *Tschau Sepp Game Component Scope Document* should be consulted.

The scope of the `LOGIC` sub-component is to simulate a Tschau Sepp game between multiple players by maintaining the game state and by enforcing the rules of the game. Issues related to how the game is shown on the screen or how the involved computers communicate in detail via network lie outside of the scope of this sub-component.

The following table explains the key terms and abbreviations used in the document:

Term	Definition
Player	A person who can interact with the game application that has been started and is not terminated.
User	A potential player of the game.
Server	Refers to the *Multiplayer Card Games* server.
Client	Refers to the whole *Tschau Sepp Game Component* that is connected to the *Multiplayer Card Games* server.
`LOGIC`	A sub-component of the *Tschau Sepp Game Component* that is responsible for maintaining the game's logic.
`GUI`	A sub-component of the *Tschau Sepp Game Component* that is responsible for displaying all the relevant information to the player and receiving his/her actions. For this, graphical icons, text boxes and buttons are used. Furthermore, this sub-component may contain plugins, such as a chat system.
`NET`	A sub-component of the *Tschau Sepp Game Component* that is responsible for sending and receiving messages between the `NET` sub-components that are situated on the other player's computers.
Master	A mode in which the `LOGIC` sub-component can operate. In this mode it is the one who hosts the binding game state and changes it according to the received players' actions. It also informs the other `LOGIC` sub-components about the current game state.
Slave	A mode in which the `LOGIC` sub-component can operate. In this mode it merely forwards the associated player's actions that it receives to the `LOGIC` sub-component in Master mode and maintains a copy of the game state.
Message	Information that travels between components and between computers.
Game Host	The computer whose `LOGIC` sub-component is in Master mode.
Player Action	Refers to an atomic act a player can make. E.g. playing exactly one card, saying an announcement, quitting the game. It does **not** stand for a set of acts that constitute a player's turn.
Associated Player	Refers to the player that is sitting on the computer that is running the instance of the `LOGIC` sub-component.
Announcement	A player action that has to be done just before the last or second last card is played. This means saying the words "Tschau" or "Sepp", respectively.
Playing stack	Represents the collection of cards where players have to play cards on top of. The cards are face-up and stacked on top of each other, with the latest played card visible on top.
Drawing stack	Represents the collection of cards where players have to draw cards from. The cards are face-down and stacked on top of each other, with the next card to be drawn on top.

1.3 References

The structure and format of this document was chosen according to the *IEEE Std 830-1998*[4] standard, as well as previous year's documents from the *DOSE*[5] course.

The information in this document is primarily based on the *Tschau Sepp Game Component Scope Document*, which was previously released by the whole group. The SRS of both the `GUI` and `NET` sub-components are described in a separate document, which was written by the team *HUT2*.

1.4 Overview

Section 2 defines the general product functions, intended application constraints to be respected and the assumption made in order to define requirements. In short, it digs further into the product specification, delineating the perspective of this product, the functions and other general information.

Section 3 lists the specific functional and non-functional requirements in detail.

Appendix A contains detailed information about the system's interfaces.

Appendix B shows the licence agreement that applies to the final product.

2 Overall description

We present an overall description of the `LOGIC` sub-component of the *Tschau Sepp Game Component*.

2.1 Product perspective

The `LOGIC` sub-component cannot work on its own but requires both the `GUI` and `NET` sub-components. However, the `LOGIC` sub-component represents the central part of the all the three sub-components that make up the entire *Tschau Sepp Game Component*.

The `LOGIC` sub-component does not directly have an interface that connects two running `LOGIC` instances. Instead each `LOGIC` sub-component is connected to a `NET` sub-component that is responsible to exchange messages between computers. The `LOGIC` sub-component, on its own, has two interfaces: one to the `GUI` sub-component and another one to the `NET` sub-component.

[4] IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications

[5] Distributed and Outsourced Software Engineering course at ETH Zürich

Any detailed definition of the other sub-components is out of scope of this document.

Figure 1 presents an overall view of the application architecture. With this we want to present the eight different interfaces provided for the four different components that form the *Tschau Sepp Game Component*. This are named starting with the letter `I` (standing for interface).

There are **no** interfaces between the *Tschau Sepp Game Component* and the *Multiplayer Card Games* server.

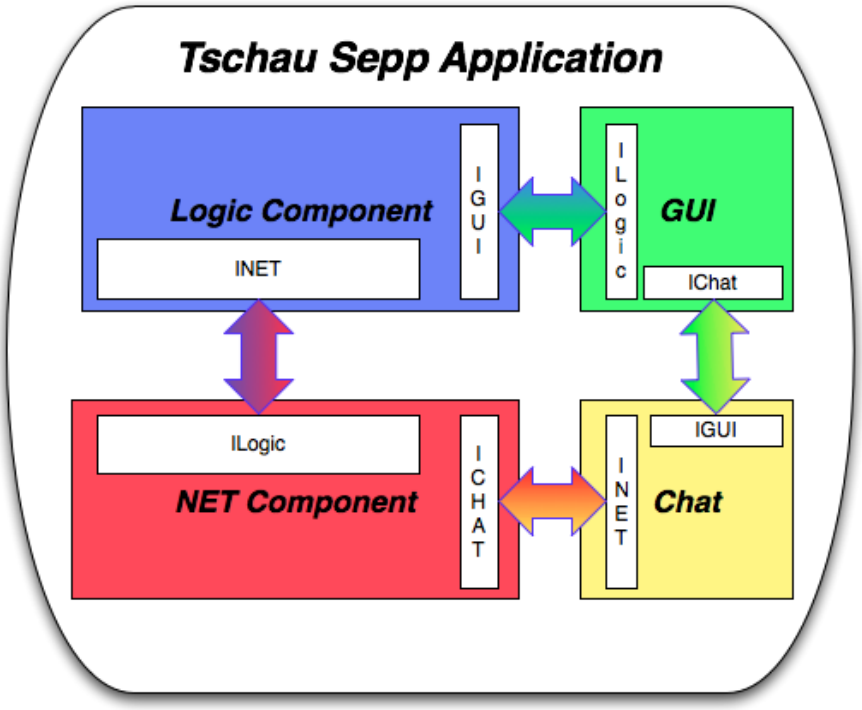

Figure 1: Exposed Interfaces between Tschau Sepp different components

2.2 Product functions

We present a general overview of all the functions that this sub-component shall provide. A more detailed explanation of the functionalities is located in section 3.

In general, the functionality of a `LOGIC` sub-component is

- to store the entire game state;
- to collect players' actions from both the `GUI` and `NET` sub-components;
- to change the game state such that the rules of the game are enfored;
- to provide the `GUI` sub-component with all the information about the game state.

More specificially, an instance of the `LOGIC` sub-component can operate in either Master or Slave mode. Of all the computers that are connected as a client to the *Multiplayer Card Games* server, exactly one has a `LOGIC` sub-component instance running that is in Master mode. This computer represents the game host.

In Master mode the `LOGIC` sub-component's functionality is to

- store the **binding** version of the entire game state;
- receive the actions of **all** players, using both the `GUI` and `NET` sub-components;
- validate received player actions, i.e. check if they are conforming to the rules of the game;
- change its game state according to all the valid player actions;
- send the changed game state to all the other `LOGIC` sub-components, which are in Slave mode, using the `NET` sub-component;
- provide the `GUI` sub-component with all the information about the game state.

In Slave mode the `LOGIC` sub-component's functionality is to

- store a copy of the entire game state;
- receive **only** the actions of the player that is associated with the `LOGIC` sub-component, using the `GUI` sub-component;
- forward a player action to the `LOGIC` sub-component that is in Master mode, using the `NET` sub-component;
- replace the game state if an updated version is received from the `LOGIC` sub-component that is in Master mode;
- provide the `GUI` sub-component with all the information about the game state.

If a player quits the game or his/her computer gets disconnected from other reasons, the game will still go on, as long as the player was not sitting on computer with the system in Master mode.
The following features will not be part of the `LOGIC` sub-component:

1. **State or game recovery.**
   The system will not support any kind of state or game recovery, in case of network failure. For example, if a computer that runs the system in Slave mode gets disconnected, it will not be able to join the running game again.

2. **Dynamic Master mode re-assignment.**
   If the computer that is running the system in Master mode is disconnected, the entire game will immediately stop. The role of the Master will not be handed over to another system.

2.3 User characteristics

We have not detected any kind of possible *User* for the `LOGIC` sub-component. Human beings do not have a direct interaction with the system.

However, there are conceptually different states in which a player can be in. These states are:

1. **Active player state.** A player that is still playing and whose turn it is. He/she has an undefined place number yet.

2. **Non-active player state.** A player that is still playing and whose turn it is not. He/she has an undefined place number yet.

3. **Observing player state.** A player that got rid of all his/her cards. He/she already has a defined place number.

Futhermore, a player can be in one of the following states:

1. **Game hosting player state.** A player that is associated with a `LOGIC` sub-component that is in Master mode. If he/she quits the game, the entire game will stop.

2. **Non-game hosting state.** A player that is associated with a `LOGIC` sub-component that is in Slave mode. If he/she quits the game, the entire game will go on.

Confer section 3.1.4 for more information on the player's possible actions.

2.4 Constraints

This document does not represent an SRS of the whole *Tschau Sepp Game Component*. It is intended to provide a concrete SRS for the `LOGIC` sub-component without taking into consideration the other sub-components that are required to run the game. This we have detected to be a constraint since it can produce ambiguities between requirements of the other components. Other constraints are related to the lack of constant communication at the time of writing the

separated SRS documents and possible different SRS standards used for producing the mentioned document. The rest of the SRS is provided in a separate document containing the SRS for the `NET` and `GUI` sub-components. All the previously described constraints can have an economically and developing time impact on the project. The reason is that developers will have to deal with ambiguities, different standards and to highlight two different documents that do not represent a complete SRS of the project.

2.5 Assumptions and dependencies

The assumptions in this document are related to the two different teams producing them. It is assume that both teams are using the same standard, both teams are following the guidelines produce and written in the *Tschau Sepp Component Scope Document*. This also establish a dependency between both teams for producing correct documentation.
Assumptions regarding a running game are:

- There is a stable connection between computers.
- There are not message corruption or errors.
- The `LOGIC` sub-component in Master mode player keeps a constant and correct state of the game.

3 Specific requirements

In the following, the `LOGIC` sub-component is referred to as the system.

Property	Description
Requirement ID	Defines a unique symbolic name for the requirement. It also reflects which functional group it belongs to.
Title	A descriptive title for the requirement.
Priority	Defines the order in which requirements should be implemented. Priorities are designated (highest to lowest) 1, 2, and 3 ... Requirements of priority 1 are mandatory for the *First Implementation*; requirements of priority 2 are mandatory for the *Final Implementation*. A priority greater or equal than 3 represents optional features.
Risk	Specifies the risk of not implementing the requirement. It shows how critical the requirement is to the system as a whole. The following risk levels are defined over the impact of not being implemented correctly. • Critical (C) It will break the main functionality of the system. The system cannot be used if this requirement is not implemented. • High (H) It will impact the main functionality of the system. Some function of the system could be inaccessible, but the system can be generally used. • Medium (M) It will impact some system features, but not the main functionality. The system can still be used with some limitation. • Low (L) The system can be used without limitation, but with some workarounds.
References	Lists the IDs of requirement that are also relevant in this context.

3.1 Functional

3.1.1 General

Req. ID	R 3.1.1.001
Title	One system per player
Description	The system shall be associated with exactly one specific player.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.1.002
Title	The game state
Description	The system shall store a complete Tschau Sepp game state, which includes the following: • An ordered list of the participating players; • The content of the drawing stack; • The content of the playing stack; • All players' cards; • All players' current place number; • The current suit to be played; • The current sense of rotation; • The currently active player; • If the player has already played a card on his/her turn; • How many cards the currently active player has to draw due to a previous played 7; • If the player has just said "Tschau" or "Sepp".
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.1.003
Title	GUI sub-component interface
Description	The system shall provide all the information that constitute the game state to the GUI sub-component.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.1.004
Title	Network communication
Description	The system shall be collaborating with other instances of the same system.
Priority	1
Risk	C
References	R 3.1.1.001

Req. ID	R 3.1.1.005
Title	System states
Description	The system shall either be in Master or Slave mode.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.1.006
Title	One master per game
Description	In a set of collaborate systems, exactly one shall be in Master mode, which makes the computer it is running on the game host.
Priority	1
Risk	C
References	R 3.1.1.005

Req. ID	R 3.1.1.007
Title	Quitting
Description	If a player that is associated with a system in Slave mode **quits** the game, he/she shall be removed from the list of players and the system he/she is associated with shall be disconnected from the system in Master mode.
Priority	2
Risk	H
References	R 3.1.1.005

3.1.2 Master mode

Req. ID	R 3.1.2.001
Title	Receive player actions
Description	If in Master mode, the system shall receive the actions **of all** participating players.
Priority	1
Risk	C
References	R 3.1.2.002, R 3.1.2.003

Req. ID	R 3.1.2.002
Title	Receive associated player's actions
Description	If in Master mode, the actions of the associated player shall be received directly from the GUI sub-component.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.2.003
Title	Receive other players' actions
Description	If in Master mode, the actions of the other players shall be received indirectly from NET sub-component.
Priority	1
Risk	C
References	R 3.1.3.001

Req. ID	R 3.1.2.004
Title	Validate players actions
Description	If in Master mode, the system shall validate any player action that has been received, in order to enforce the rules of the game.
Priority	2
Risk	H
References	R 3.1.5.001 - R 3.1.5.012

Req. ID	R 3.1.2.005
Title	Update game state
Description	If in Master mode, the system shall change the game state if a received player action has been successfully validated, as to reflect what the action entails.
Priority	1
Risk	C
References	R 3.1.1.002, R 3.1.2.004

Req. ID	R 3.1.2.006
Title	Distribute game state
Description	If in Master mode, when the game state has been changed, the system shall inform all connected systems, which are in Slave mode, about the new game state, and thereby confirm that the action was valid.
Priority	1
Risk	C
References	R 3.1.1.004, R 3.1.3.005

Req. ID	R 3.1.2.007
Title	Invalid actions
Description	If in Master mode, when the system receives an invalid action, it shall inform the system where the action came from about the fact as well as the reason why the action was invalid.
Priority	2
Risk	H
References	R 3.1.2.004

Req. ID	R 3.1.2.008
Title	Game Restart
Description	If in Master mode, when all the players in the list of participating players have a place number defined, the system shall wait for exactly 10000 milliseconds and set the game state to the initial one. This requirement is regarded as an alternative to the stopping criterion described in R 3.1.10.001.
Priority	3
Risk	L
References	R 3.1.10.001

Req. ID	R 3.1.2.009
Title	Kicking Players
Description	If in Master mode, when the system receives the action **kick** (which has a parameter that specifies a player) it shall remove the player in question from the list of players.
Priority	3
Risk	L
References	NONE

3.1.3 Slave mode

Req. ID	R 3.1.3.001
Title	Forward current player actions
Description	If in Slave mode, the system shall receive only the actions of the player associated with it from the GUI sub-component and forward those actions to the system which is in Master mode through the NET sub-component.
Priority	1
Risk	C
References	R 3.1.2.003

Req. ID	R 3.1.3.002
Title	Validate current player actions
Description	If in Slave mode, the system shall only forward actions that it has validated itself to the system in Master mode.
Priority	3
Risk	M
References	R 3.1.5.001 - R 3.1.5.012

Req. ID	R 3.1.3.003
Title	Invalid player actions
Description	If in Slave mode, when the system receives a message from the system in Master mode that the last player action was invalid, it shall store and provide the reason to the GUI sub-component.
Priority	3
Risk	M
References	R 3.1.2.004, R 3.1.2.007

Req. ID	R 3.1.3.004
Title	Game state integrity
Description	If in Slave mode, the system shall not directly modify its game state.
Priority	1
Risk	C
References	R 3.1.3.005

Req. ID	R 3.1.3.005
Title	Game state updates
Description	If in Slave mode, the system shall receive updated game states from the system in Master mode. These updated game states shall be an identical clone of the game state maintained on the Master.
Priority	1
Risk	C
References	R 3.1.2.006

3.1.4 Initial Game State

Req. ID	R 3.1.4.001
Title	Cards distribution
Description	The initial distribution of cards shall be as follows: all players have five cards; the playing stack consists of one card; the drawing stack consists of all the remaining cards. All of the cards are randomly chosen.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.4.002
Title	Initial place numbers
Description	The place numbers of all the players in the list of participating players, shall initially have the special value **undefined**.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.4.003
Title	Initial sense of rotation
Description	The initial sense of rotation shall be clockwise unless the initial card on the playing stack is a 10, in which case it is counter-clockwise.
Priority	2
Risk	H
References	R 3.1.4.001

Req. ID	R 3.1.4.004
Title	Initial suit
Description	The initial suit to be played shall be the same as the card on the playing stack. (Even for a Jack)
Priority	1
Risk	C
References	R 3.1.4.001

Req. ID	R 3.1.4.005
Title	Initial draw
Description	The initial amount of cards to be drawn due to a 7 is 0, unless the top card on the playing stack is a 7, in which case the amount shall be 2.
Priority	2
Risk	H
References	R 3.1.4.001

Req. ID	R 3.1.4.006
Title	Initial active player
Description	The player that is initially active shall be chosen depending on the top card on the playing stack as follows: If the card is an 8 the third player on the list shall be active. If the card is a 10 the last player on the list shall be active. In any other case the second player on the list shall be active.
Priority	2
Risk	H
References	NONE

3.1.5 Rules of the game

Req. ID	R 3.1.5.001
Title	Game card deck.
Description	There shall always be exactly 36 cards involved. Each of them has one of the following four suits: Spades, Diamonds, Clubs or Hearts, as well as one of the following values: Ace, 6, 7, 8, 9, 10, Jack, Queen or King.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.5.002
Title	Player possible actions.
Description	A player's action shall be one of the following: • to draw exactly one card from the drawing stack; • to put exactly one card from the player's set of cards on top of the playing stack; • to choose a suit; • to pass; • to say "Tschau"; • to say "Sepp"; • to **quit** the game. • to **kick** a player. (optionally)
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.5.003
Title	Action allowance.
Description	A player shall only be allowed to perform an action if he/she is active. The only exception is the action of **quitting** the game, which is valid at any time.
Priority	1
Risk	C
References	R 3.1.5.002

Req. ID	R 3.1.5.004
Title	Playable card.
Description	A player shall be able to put one of his/her cards on the top of the playing stack if this card has either the same value as the one on top of the playing stack or the suit matches the current suit to be played. Furthermore, if the card is a Jack, it can always be played.
Priority	1
Risk	M
References	NONE

Req. ID	R 3.1.5.005
Title	Play card by suit type.
Description	If the top card of the playing stack is not a Jack, the current suit to be played shall be the suit of the top card.
Priority	1
Risk	H
References	R 3.1.5.004

Req. ID	R 3.1.5.006
Title	Draw card allowance.
Description	A player shall be able to draw once a card while he/she is active if he/she cannot or doesn't want to play a card yet.
Priority	1
Risk	C
References	R 3.1.5.008

Req. ID	R 3.1.5.007
Title	Draw card and Pass.
Description	A player shall be able to **pass** if he/she has already drawn a card while he/she was active and still cannot or doesn't want to play a card.
Priority	2
Risk	M
References	NONE

Req. ID	R 3.1.5.008
Title	Active to inactive player state.
Description	A player shall be no more active just after having played a card different from an Ace or a Jack; or after having chosen the suit to be played; or after having **passed**.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.5.009
Title	Next player turn is?
Description	If a player is not active anymore, the next active player shall be chosen from the list of all players as follows: One starts at the position of the currently active player. If the current sense of rotation is clockwise, one goes through the list from top to bottom, otherwise from bottom to top. The list is considered to be circular. If the top card on the playing stack is not an 8, the next player that has an **undefined** place number is chosen. If the top card is an 8, the second next player that has an **undefined** place number is chosen (i.e. one player with **undefined** place number is skipped).
Priority	1
Risk	C
References	R 3.1.5.008

Req. ID	R 3.1.5.010
Title	Play Jack and choose suit.
Description	A player shall be obliged to freely chose the current suit to be played if and only if he/she has just played a Jack. (The actual suit of the Jack is irrelevant)
Priority	2
Risk	M
References	NONE

Req. ID	R 3.1.5.011
Title	Play 10 change rotation.
Description	If a player plays a 10, the current sense of rotation shall be changed between clockwise and counter-clockwise.
Priority	2
Risk	M
References	NONE

Req. ID	R 3.1.5.012
Title	Ace on top of stack and draw card.
Description	If the top card on the playing stack is an Ace and the active player cannot play a card on top of it he/she shall be able to draw a new card.
Priority	2
Risk	M
References	NONE

3.1.6 Drawing Cards Due to A 7

Req. ID	R 3.1.06.001
Title	Increment draw card amount.
Description	If a player plays a 7, the amount of cards to be drawn due to a 7 shall rise by 2.
Priority	2
Risk	M
References	NONE

Req. ID	R 3.1.06.002
Title	Player draw card or play 7.
Description	If the amount of cards to be drawn due to a 7 is non-zero, the player shall be obliged to either draw this amount of cards from the drawing stack or to play a 7. Drawing cards due to a 7 does not forfeit the right to draw one more card afterwards.
Priority	2
Risk	M
References	R 3.1.06.001

Req. ID	R 3.1.06.003
Title	Reset draw card amount
Description	If a player has drawn the amount of cards he/she was obliged to, the amount of cards to be drawn due to a 7 shall be reset to 0.
Priority	2
Risk	M
References	NONE

3.1.7 Announcements

Req. ID	R 3.1.07.001
Title	Say Tschau
Description	A player shall be obliged to **say Tschau** before playing his/her second last card.
Priority	2
Risk	H
References	NONE

Req. ID	R 3.1.07.002
Title	Say Sepp
Description	A player shall be obliged to **say Sepp** before playing his/her last card.
Priority	2
Risk	H
References	NONE

Req. ID	R 3.1.07.003
Title	Did Not say Tschau or Sepp?
Description	A player shall receive the top card from the drawing stack automatically if he/she has forgotten to say either "Tschau" or "Sepp", when he/she had to.
Priority	2
Risk	H
References	R 3.1.07.001, R 3.1.07.002

3.1.8 Place Numbers

Req. ID	R 3.1.08.001
Title	Assigning number to player
Description	A player shall be assigned place number the game if he/she does not have anymore cards. The place number shall be the lowest positive integer number that has not already been assigned to any player.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.08.002
Title	One player numbering check
Description	If the list of participating players contains only one player that has **undefined** as a place number, this value shall be changed automatically to the lowest positive integer number that has not already been assigned to any player.
Priority	1
Risk	C
References	R 3.1.08.001

3.1.9 Drawing Stack Restocking

Req. ID	R 3.1.09.001
Title	Player quit restock cards in deck stack
Description	If a player that is associated with a system in Slave mode has **quit** the game his/her cards shall be added to the drawing stack in a random fashion.
Priority	2
Risk	C
References	NONE

Req. ID	R 3.1.09.002
Title	Re-shuffle deck stack when empty.
Description	If the drawing stack consists of only one card, all but the top card from the playing stack shall be added to the drawing stack in a random fashion.
Priority	2
Risk	C
References	NONE

3.1.10 Stopping Criteria

Req. ID	R 3.1.10.001
Title	Stop condition one.
Description	If the list of participating players contains no player that has **undefined** as place number, the system shall stop, as the game is considered to be finished.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.1.10.002
Title	Stop condition two.
Description	If the player that is associated with the system in Master mode **quits** the game, the system shall stop.
Priority	1
Risk	H
References	NONE

3.2 Non-Functional

Req. ID	R 3.2.001
Title	Availability
Description	The system will be available to the NET and GUI sub-components as long as the system is running.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.2.002
Title	Reliability
Description	The system will always work correctly and uncorrupted, given its input from the GUI and NET sub-components.
Priority	1
Risk	C
References	NONE

Req. ID	R 3.2.003
Title	Integrity
Description	The system will maintain information integrity; the Slaves may use an older version of the Game State, but as soon as they receive an update, they shall act upon it, so that the state is updated at most 2 minutes after the Master's Game State was updated.
Priority	1
Risk	H
References	NONE

Req. ID	R 3.2.004
Title	Robustness
Description	The system shall not recover from error states produced by external factors.
Priority	1
Risk	L
References	NONE

Req. ID	R 3.2.005
Title	Performance
Description	The system shall process a notification from the NET or GUI sub-components in at most 1000 milliseconds.
Priority	2
Risk	L
References	NONE

Req. ID	R 3.2.006
Title	Maintainability
Description	The system's source code shall follow these rules: • Good indentation is required. • Each variable should be named in a suggestive manner. • Each name given to a class or a feature has to clearly identify its meaning and suggest its behavior. Comments should be present to clarify meanings when names do not suffice. • The Class names have to be prefixed with TS_
Priority	2
Risk	L
References	NONE

A System Interfaces

Our system was designed, having in mind a very simple API. The application's sub-components communicate mainly using a message driven system. The latter allows us to have a simple and decoupled application. The three sub-components are represented by the classes `TS_NET`, `TS_GUI` and `TS_LOGIC`. In order to pass messages, those classes use dedicated features. `TS_LOGIC` and `TS_GUI` use a feature called `get_message`. `TS_NET` uses the features broadcast_message and `send_message_to`.

On top of that, the main classes have several additional features that are necessary for the communication, but which do not fit into the message passing paradigm.

In the following, all the features that are relevant for the interface are listed and described. For every featured the pre- and postcondition is shown. At the end of the lists, we also present the class invariants.

A.1 The `TS_LOGIC` Class

`get_message (m: TS_MESSAGE)`	
Require	`is_draw_card_action (m) or is_play_card_action (m) or is_choose_suit_action (m) or is_pass_action (m) or is_say_tschau_action (m) or is_say_sepp_action (m) or is_quit_action (m) or is_kick_action (m) or is_invalid_action_notification(m)`
Ensure	`True`
Description	Receives and analyzes messages sent by the `GUI` and `NET` sub-components.

`is_master: BOOLEAN`	
Require	`True`
Ensure	`True`
Description	Returns whether or not this system is running under master mode.

`add_new_player (user_id: STRING_8; user_ip: STRING_8)`	

Require	`is_master = True`
	`user_id /= Void and then not user_id.is_empty`
	`user_ip /= Void and then not user_ip.is_empty`
	`connected_players.count < Maximum_player_count`

Ensure	`connected_players.count = old connected_players.count + 1`
Description	Adds a new player to the game.

connected_players: LIST [TS_PLAYER]	
Require	True
Ensure	Result /= Void
Description	Return a list of all connected players.

disconnect (user_id: STRING_8)	
Require	is_master = True user_id /= Void and then not user_id.is_empty connected_players.count > 0
Ensure	connected_players.count = old connected_players.count - 1
Description	Disconnect a player with ID user_id from the game.

associated_player: TS_PLAYER	
Require	True
Ensure	Result /= Void
Description	Returns the player that is associated with this instance

Class Invariant
game_is_running implies (connected_players.count ≥ Minimum_player_count and connected_players.count ≤ Maximum_playser_count)

A.2 The TS_GUI Class

A.2.1 Interact with LOGIC sub-component

get_message (m: TS_MESSAGE)	
Require	is_draw_card_action (m) or is_play_card_action (m) or is_choose_suit_action (m) or is_pass_action (m) or is_say_tschau_action (m) or is_say_sepp_action (m) or is_quit_action (m) or is_kick_action (m) or is_invalid_action_notification(m)
Ensure	True
Description	Receives and analyzes messages sent by the LOGIC sub-component; if there is any change in game state, this procedure will call re-draw_stage procedure

A.2.2 User interaction handler

click_card (card: TS_CARD)	
Require	is_draw_card_action (m) or is_play_card_action (m)
Ensure	True
Description	Pass a card-clicking event to LOGIC subcomponents to valid.

choose_suit (suit: TS_SUIT)	
Require	is_choose_suit_action
Ensure	True
Description	Choose a suit for the next player to play.

say_sepp	
Require	is_say_sepp_action
Ensure	True
Description	Say sepp button handler.

say_tschau	
Require	is_say_tschau_action
Ensure	True
Description	Say tschau button handler.

kick_player (slave_player: TS_PLAYER)	
Require	associated_player.is_master, is_kick_action
Ensure	connected players.count = old connected players.count-1
Description	Kick player button handler, this button is visible only in master mode.

quit_game	
Require	is_quit_action
Ensure	True
Description	Quit button handler.

pass	
Require	is_pass_action
Ensure	True
Description	Pass button handler.

A.2.3 Change the visualization

init_game_stage	
Require	– is the first procedure to be called
Ensure	True
Description	Initilize the game stage.

redraw_stage	
Require	is_invalid_action_notification = FALSE
Ensure	True
Description	Redraw the game visualiztion (cards, player list) after a valid notification is passed to TS_GUI from TS_LOGIC or when there is a change in player list.

show_current_rotation	
Require	–
Ensure	is_rotate_action
Description	Show the current sense of rotation.

show_kick_button	
Require	–
Ensure	associated_player.is_master
Description	Show the kick button for master player.

set_current_effect_card(card: TS_CARD)	
Require	–
Ensure	is_draw_card_action
Description	Show the current effect card.

play_card(played_card: TS_CARD)	
Require	–
Ensure	is_play_card_action
Description	Move the card from player's hand to the playing stack, and redraw stage.

draw_card(drawed_card: TS_CARD)	
Require	–
Ensure	is_draw_card_action
Description	Move the card from drawing stack to player's hand , and redraw stage.

notify_sepp	
Require	–
Ensure	is_say_sepp_action
Description	Notify when a player says sepp .

notify_tschau	
Require	–
Ensure	is_say_tschau_action
Description	Notify when a player says tschau .

A.2.4 Attribute

list_card: LIST[TS_CARD]	
Require	–
Ensure	number_of_card = 36
Description	List of card .

connected_players: LIST[TS_PLAYER]	
Require	–
Ensure	Result /= Void
Description	List of player .

pass: BUTTON	
Require	–
Ensure	–
Description	Pass button .

quit: BUTTON	
Require	–
Ensure	–
Description	Quit button .

quit: BUTTON	
Require	–
Ensure	–
Description	Kick button .

associated_user: TS_PLAYER	
Require	TRUE
Ensure	Result /= Void
Description	Returns the player that is associated with this instance.

A.3 The TS_NET Class

is_master: BOOLEAN	
Require	TRUE
Ensure	Result /= Void
Description	TRUE if this is a master player and FALSE if not .

port: INTEGER	
Require	TRUE
Ensure	port $\geq$ 1029 and port $\leq$ 65535
Description	A number indicate the port to communicate between program instances.

associated_user: TS_PLAYER	
Require	TRUE
Ensure	Result /= Void
Description	Returns the player that is associated with this instance.

listener: NETWORD_DIAGRAM_SOCKET	
Require	TRUE
Ensure	Result /= Void
Description	Returns the listener.

broadcast_message: TS_MESSAGE	
Require	TRUE
Ensure	Result /= Void
Description	Send the message to all slave players.

start_server	
Require	is_master
Ensure	–
Description	Start server.

stop_server	
Require	is_master
Ensure	–
Description	Stop server.

disconnect_user(user: TS_USER)	
Require	is_master
Ensure	–
Description	Disconnect an user (kick or no more network activity).

invite(user: TS_USER)	
Require	is_master
Ensure	–
Description	Invite a player.

connect_to(server_ip: STRING, message: TS_MESSAGE)	
Require	–
Ensure	–
Description	Connect to a server.

disconnect(server_ip: STRING, message: TS_MESSAGE)	
Require	–
Ensure	–
Description	Disconnect and quit game.

send_message_to(player: TS_PLAYER,message: TS_MESSAGE)	
Require	–
Ensure	–
Description	Send messages to server about the action update, chat, etc; and server will response.

A.4 Message Passing Diagram

In the following diagram, a player that is asscociated with a system in Slave mode decides to draw a card from the drawing stack. It is shown how the messages will be passed between the different sub-component instances.

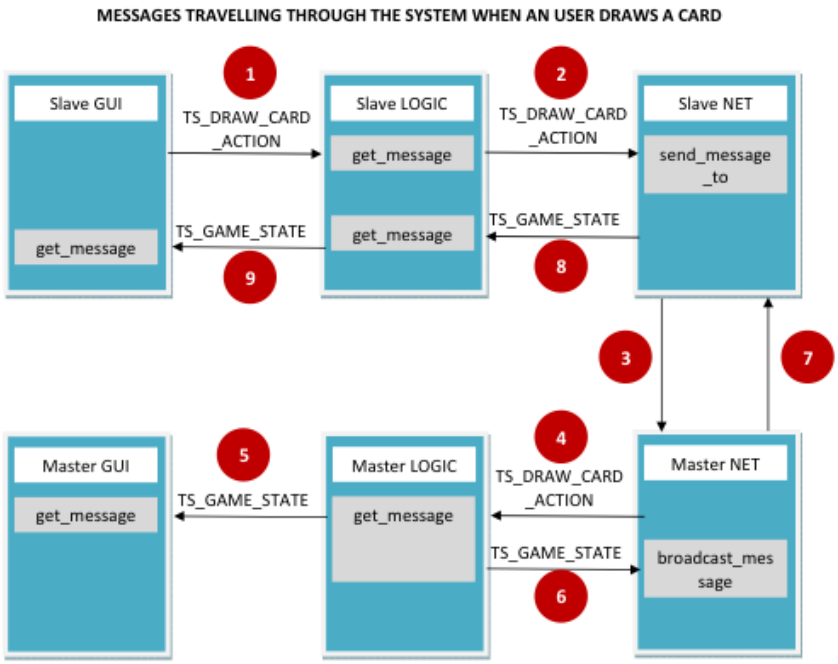

Figure 2: Ordered message flow picture.

B License Agreement